

NGUYỄN XUÂN HUY

CHUYÊN TIN 11

CÁC CHUYÊN ĐỀ CHỌN LỌC VỚI NGÔN NGỮ C++
LÝ THUYẾT, BÀI TẬP VÀ LỜI GIẢI



NHÀ XUẤT BẢN THÔNG TIN VÀ TRUYỀN THÔNG

Số xác nhận đăng ký xuất bản: 2856-2025/CXBIPH/3-119/TTTT

Số quyết định xuất bản: 725/QĐ-NXB TTTT ngày 03 tháng 9 năm 2025

Nộp lưu chiếu: Quý IV năm 2025

ISBN: 978-604-45-1537-3

NGUYỄN XUÂN HUY

CHUYÊN TIN 11

CÁC CHUYÊN ĐỀ CHỌN LỌC VỚI NGÔN NGỮ C++

LÝ THUYẾT, BÀI TẬP VÀ LỜI GIẢI

NHÀ XUẤT BẢN THÔNG TIN VÀ TRUYỀN THÔNG

MỤC LỤC

<i>Lời nói đầu</i>	7
CHƯƠNG 1. ADVANCED DATA STRUCTURES	9
Programs = Algorithms + Data Structures	9
C++ 11	10
Vector	10
Set	21
List	26
Stack	28
Queue	49
Deque	52
Map	58
Words	60
Farey	64
Luyện tập Farey	73
Bài giải Farey	73
Class Frac	75
OverLoad	75
Implement of Frac Class	76
Frac Testing 1	80
Frac Testing 2	81
Giải thích thiết kế và các nguyên lý Test	82
Frac Test 3	85
CHƯƠNG 2. GRAPH	87
Find-Union	88
Connected Zones	94

Connectivity	97
Spanning Tree.....	98
Minimal Spanning Tree	101
Spanning Forest	104
Minimal Spanning Forest	113
Cycle.....	114
Cycle In Connected Zone	117
Cycles in Spanning Forest	118
LAN.....	129
Water Valves	132
Bridges.....	134
Intersection Of Spanning Trees	148
Joint Vertex	149
Classroom LAN.....	154
IT Teams.....	157
Ants.....	160
Subgraph.....	160
Werewolf	165
CHƯƠNG 3. ADVANCED SORTING	178
Merge Sort	178
Topo sort.....	180
Tortoise.....	185
Floor Decoration.....	189
CHƯƠNG 4. ỨNG DỤNG CÁC CHIẾN LƯỢC THIẾT KẾ	
THUẬT TOÁN NÂNG CAO	194
Flycam	194
Philosophers	198
Clocks	206
Unique Number	212
Lập lịch cho hai máy	214

Nearest Points	218
Longest Common Substring	225
Longest Monotone Segment.....	229
Longest Repeated Sliding Segments	232
Dictionary	239

CHƯƠNG 5. THIẾT KẾ VÀ PHÁT TRIỂN MỘT SỐ SẢN PHẨM

PHẦN MỀM ỨNG DỤNG	244
Sudoku	244
Các biến thể của Sudoku	258
Game Sudoku	264
TÀI LIỆU THAM KHẢO	279

LỜI NÓI ĐẦU

Tiếp nối cuốn sách "Chuyên Tin 10: Các chuyên đề chọn lọc với ngôn ngữ C++ (Lý thuyết, bài tập và lời giải)", cuốn sách "*Chuyên Tin 11: Các chuyên đề chọn lọc với ngôn ngữ C++ (Lý thuyết, bài tập và lời giải)*" được biên soạn nhằm cung cấp các chuyên đề chọn lọc với các bài Toán Tin nâng cao cho học sinh các khối chuyên lớp 11, định hướng tới những kỹ thuật lập trình cơ bản để giải những bài toán khó trên máy tính.

Nội dung sách được biên soạn theo:

KHUNG CÁC CHUYÊN ĐỀ MÔN TIN HỌC

DÀNH CHO TRƯỜNG TRUNG HỌC PHỔ THÔNG CHUYÊN

Một bài toán tin được hiểu là khó nếu ta sử dụng thuật giải mới nảy sinh trong đầu khi vừa biết nội dung bài toán thì ta thường thu được kết quả sai, hoặc là lời giải nhận được sẽ không hữu hiệu theo nghĩa chương trình đòi hỏi quá nhiều bộ nhớ hoặc/và chạy quá lâu. Những thuật giải nảy sinh ngay trong đầu như vậy thường được gọi là "thuật giải tự nhiên". Dĩ nhiên, khái niệm này chỉ là tương đối. Nếu bạn đã nắm vững nhiều dạng thuật giải và đã từng thử sức với nhiều bài toán khó thì đến một lúc nào đó, các thuật giải tự nhiên của bạn sẽ đáng tin cậy. Đó cũng chính là mục đích của sự học tập và rèn luyện và cũng là ước mơ của người viết tập sách này.

Việc đọc và tiếp cận cuốn sách không đòi hỏi bạn đọc có nền tảng kiến thức chuyên sâu. Chỉ cần bạn đã từng làm quen với một trong các ngôn ngữ lập trình phổ biến như C, Pascal, Java hoặc Python, bạn hoàn toàn có thể hiểu và cùng đóng góp ý kiến để nội dung ngày một hoàn thiện hơn.

Các kỹ thuật lập trình được minh họa qua những bài toán cụ thể tương đương với trình độ nâng cao của học sinh và sinh viên, đặc biệt là bám sát Chương trình Tin học cho các khối chuyên Tin của trường chuyên. Hình thức phát biểu bài toán suy cho cùng là không quan trọng. Các kỹ thuật lập trình và phương pháp xây dựng thuật giải cho những bài toán thường được dùng rộng rãi trong quá trình thiết kế và cài đặt các phần mềm ứng dụng trong thực tiễn, cho nên việc sớm làm chủ các tri thức này mới thật sự là cần thiết. Nội dung sách phục vụ đối tượng chính là học sinh chuyên Tin, tuy nhiên những kỹ thuật và thuật toán trong sách cũng hữu ích cho sinh viên muốn nâng cao năng lực lập trình và giải thuật.

Nội dung sách gồm hai phần. Phần thứ nhất giới thiệu vắn tắt về bản chất các kiểu dữ liệu, các phương pháp và kỹ thuật lập trình và các đề toán để các bạn thử sức. Phần thứ hai trình bày và phân tích chi tiết lời giải cùng với những bình luận và xuất xứ của các bài toán.

Cuốn sách này cũng cung cấp toàn văn các chương trình viết bằng ngôn ngữ lập trình C++ để bạn đọc tiện so sánh với lời giải của mình.

Những nội dung trong tập sách được tập hợp và chỉnh lý từ các bài giảng về thuật toán và lập trình, từ các cuốn sách và các bài viết của tác giả đăng trong tạp chí Tin học và Nhà trường và một số lời giải hay của các bạn học sinh.

Hy vọng rằng trong các dịp khác tác giả sẽ cung cấp thêm các phương án giải với bạn đọc. Tuy nhiên, suy cho cùng, môi trường lập trình chỉ mang tính minh họa. Khi đã biết thuật toán, việc thể hiện thuật toán đó trong môi trường lập trình cụ thể chắc chắn là việc làm quen thuộc của bạn đọc.

Bạn đọc có thể tải và cài đặt môi trường lập trình miễn phí mã nguồn mở C++ và Python tại các địa chỉ Internet dưới đây:

<http://www.bloodshed.net/index.html>

<http://wingware.com/downloads/wing-101>

Theo thông báo mới nhất của Hội Tin học Mỹ và Quốc tế thì ngôn ngữ lập trình Python được xem là ngôn ngữ cho tương lai với các đặc điểm sau:

- Cú pháp đơn giản, dễ học
- Tự động quản lý cấp phát và thu hồi miền nhớ
- Phù hợp cho phát triển các hệ thống thông minh (AI)

Thiết nghĩ rằng, từ năm nay trở đi, các trường chuyên sẽ dạy song song C++ và Python.

Một số đề bài được biên soạn lại nhằm tăng tính hấp dẫn và gần gũi với học sinh, mặc dù xuất xứ của bài được lấy từ tài liệu trích dẫn. Mong được bạn đọc lượng thứ về sự thay đổi này.

Xin được chân thành cảm ơn các thầy cô giáo, các em học sinh, sinh viên, đồng nghiệp và bạn bè đã chia sẻ kinh nghiệm và trợ giúp tài liệu, góp ý trong quá trình hoàn thành cuốn sách và rất mong tiếp tục nhận được ý kiến đóng góp để các tập sau ngày càng hoàn thiện hơn.

Tác giả

Nguyễn Xuân Huy

Chương 1

ADVANCED DATA STRUCTURES

Programs = Algorithms + Data Structures



Niklaus Emil Wirth
(15/02/1934-01/01/2024)

Đây là công thức do Niklaus Wirth, tác giả của ngôn ngữ lập trình Pascal đề xuất vào năm 1976 khi ông xuất bản cuốn sách cùng tên thể hiện những nguyên lý lập trình. Gặp những bài toán tin thực tiễn, chúng ta thường triển khai qua các pha sau đây:

1. **Đọc hiểu.** Hiểu tường tận nội dung và ý nghĩa của bài toán. Ghi rõ input và output của bài toán.
2. **Đặc tả.** Dùng ngôn ngữ tự nhiên hoặc tốt nhất là ngôn ngữ toán học diễn tả lại các yêu cầu của quá trình biến đổi input thành output. Quá trình này có thể bao gồm nhiều giai đoạn, mỗi giai đoạn giải quyết một vấn đề cụ thể.
3. Xây dựng **giải thuật**. Bạn có thể chọn một giải thuật sẵn có hoặc sửa đổi, cải tiến một giải thuật đã biết, hoặc cao hơn nữa là xây dựng một giải thuật mới để giải bài toán.
4. Lựa chọn **cấu trúc dữ liệu** cho phù hợp với thực tiễn, dễ biểu diễn và xử lý trên máy tính.
5. **Lập trình.** Viết chương trình bằng một ngôn ngữ lập trình yêu thích.
6. **Test.** Thực hiện chương trình trên tập dữ liệu thực hoặc dữ liệu giả định nhằm bao quát mọi tình huống có thể xảy ra trong thực tiễn ứng dụng của chương trình.
7. **Cải tiến.** Làm tốt chương trình theo hướng nâng cao hiệu quả: giảm miền nhớ, giảm thời gian thực hiện chương trình.

Công thức gọn gàng do Niklaus Wirth đề xuất nhắc nhở chúng ta rằng nếu có thuật toán tốt và xây dựng được cấu trúc dữ liệu phù hợp thì bạn sẽ có một chương trình (máy tính) tuyệt hảo. Điều này cũng tương tự như:

Bữa cơm ngon = Biết cách nấu ăn + Lựa chọn thực phẩm.

Chương này sẽ đề xuất một vài **kiểu dữ liệu nâng cao** phục vụ cho việc giải các bài toán tin đa dạng.

10 CHUYÊN TIN 11: CÁC CHUYÊN ĐỀ CHỌN LỌC VỚI NGÔN NGỮ C++

C++ 11

Phiên bản C++ 11 có thêm một vài cú pháp và hàm tiện ích giúp bạn lập trình. Ví dụ, để thực hiện hàm $F(x)$ trên các phần tử e của một dãy đối tượng trong p bạn có thể viết:

```
for(auto e:p) F(e);
```

Để kích hoạt phiên bản này bạn không cần cài đặt lại hệ thống mà chỉ cần vào đường dẫn sau:

```
Tools → compile options → settings → code generation → language → standard
```

Sau đó bạn chọn dòng cuối cùng của standard là GNU C++ 11 là hoàn tất.

Vector

Kiểu vector hoạt động giống như kiểu array ngoài một lợi thế là không cần đặt trước kích thước (size) của biến vector. Kích thước vector sẽ tự động co giãn tùy theo số phần tử có trong vector. Các nhà lập trình C++ chuyên nghiệp thường có thiên hướng dùng **vector**.

Khai báo kiểu vector

```
typedef vector<T> VT;  
// T là kiểu các phần tử của vector
```

Các ví dụ sau đây thực hiện trên vector `int`

Khai báo kiểu

```
typedef vector<int> VI;  
typedef vector<VI> VVI;
```

VVI là kiểu vector của VI, tức là mỗi phần tử của VVI lại là một vector `int`.

Khai báo biến vector

Ví dụ

```
VI v;  
VVI vv;
```

Nạp trị cho vector

+ *Nạp trực tiếp qua hàm `push_back`*

```
VI v; // v = [] (empty)  
for(int i = 1; i < 4; ++i)  
    v.push_back(2*i+1);  
// v = [3, 5, 7]
```

+ *Nạp trực tiếp*

```
VI v = {3, 5, 7};
```

+ *Nạp từ mảng (copy array → vector)*

```
int a[] = {3, 5, 7};
VI v;
for(int x:a)
    v.push_back(x); // v = [3, 5, 7]
```

+ *Nạp trị từ vector khác (copy vector → vector)*

```
VI v; // v = [] (empty)
v.push_back(3);
v.push_back(5);
v.push_back(7);
// v = [3, 5, 7]
VI u(v); // u = [3, 5, 7]
```

+ *Nạp n trị giống nhau*

Hàm `v.assign(n, val)` nạp n trị val cho vector v.

```
int n = 5;
VI v;
v.assign(n, 2); // v = [2, 2, 2, 2, 2]
```

Truy cập vector theo index

Các chỉ số của vector biến thiên từ 0 đến `size()-1`. Ví dụ:

```
#include<bits/stdc++.h>
using namespace std;
typedef vector<int> VI;
// hiển thị vector v kèm chu thích msg
void Print(VI v, const char * msg = "") {
    cout << msg;
    for(int i = 0; i < v.size(); ++i)
        cout << " " << v[i];
}
int main() {
    VI v;
    for(int i = 1; i < 4; ++i)
        v.push_back(2*i+1);
    Print(v, "\n Init: ");
    cout << "\n T h e   E n d";
    return 0;
}
```

12 CHUYÊN TIN 11: CÁC CHUYÊN ĐỀ CHỌN LỌC VỚI NGÔN NGỮ C++

Output

```
Init:  3 5 7
T h e    E n d
```

Truy cập vector theo con trỏ

Mỗi biến vector có một con trỏ (iterator) dùng để duyệt biến đó. Nếu *i* là biến trỏ thì **i* sẽ là trị của biến trỏ *i*.

Ví dụ

```
#include<bits/stdc++.h>
using namespace std;
typedef vector<int> VI;
typedef VI::iterator IVI;
void Print(VI v, const char * msg = "") {
    cout << msg;
    for(IVI i = v.begin(); i < v.end(); ++i)
        cout << " " << *i;
}
inline void Go(const char * msg = " ? ") {
    cout << msg;
    fflush(stdin);
    if (cin.get() == '.') exit(0);
}
int main() {
    VI v;
    for(int i = 1; i < 4; ++i)
        v.push_back(2*i+1);
    Print(v, "\n Init: ");
    cout << "\n T h e    E n d";
    return 0;
}
```

Output

```
Init:  3 5 7
T h e    E n d
```

Process exited after 0.009635 seconds with return value 0

Các ví dụ về truy cập

```
v[i] = x; // phần tử thứ i
v.at(i); // phần tử thứ i
```

```

*v.begin()+i; // phần tử thứ i
*v.front(); // phần tử đầu dãy v[0]
v.back(); // phần tử cuối dãy v[v.size()-1]
*v.end(); // phần tử cuối dãy v[v.size()-1]

```

Thực hiện thao tác F trên các phần tử của vector v

Phương pháp 1: theo index

```

for(int i = 0; i < v.size(); ++i)
    F(v[i]);

```

Phương pháp 2: theo iterator

```

for(IVI i = v.begin(); i < v.end(); ++i)
    F(*i);

```

Phương pháp 3: theo phần tử

```

for(int e:v)
    F(e);

```

Từ version 11 trở đi có thể viết

```

for(auto e:v)
    F(e);

```

count

Số lần xuất hiện trị t trong vector v

```
int count(v.begin(), v.end(), t);
```

Nếu count == 0 tức là trị t không xuất hiện trong v.

find()

Muốn xác định vị trí i đầu tiên xuất hiện trị t trong vector v ta gọi hàm:

```
int i = find(v.begin(), v.end(), t) - v.begin();
```

Hàm find sẽ duyệt từ đầu (v.begin()) đến cuối (v.end()) vector v để tìm vị trí i trở đến phần tử của v mang trị t. Nếu i == v.size() tức là t không xuất hiện trong v. Ta hiểu v.end() trở đến phần tử sát sau phần tử cuối, tức là v[v.size()].

erase()

Xóa phần tử trong vector v. Sau khi xóa, v sẽ được thu gọn lại

```

v.erase(v.begin() + 10); // xóa phần tử thứ 10
v.erase(v.begin(), v.begin()+6); // xóa 6 phần tử đầu tiên
v.erase(v.begin()); // xóa phần tử đầu v[0]
v.erase(v.end()-1); // xóa phần tử cuối cùng
v.clear(); // xóa hết v = [] (empty)
resize(int m)
v.resize(m);

```

14 CHUYÊN TIN 11: CÁC CHUYÊN ĐỀ CHỌN LỌC VỚI NGÔN NGỮ C++

Chỉnh lại kích thước của vector v

Nếu $m < v.size()$ thì các phần tử từ $m+1$ trở đi sẽ bị xóa; size mới sẽ là m .

Nếu $m > v.size()$ thì các phần tử sẽ được thêm vào cuối dãy và nhận trị 0; size mới sẽ là m .

empty()

Kiểm tra v rỗng ? Kết quả: true nếu v rỗng; false nếu $v.size() > 0$.

`v.empty();`

pop_back()

loại bỏ phần tử cuối khỏi vector v , size giảm 1.

`void v.pop_back(); // v.erase(v.end()-1)`

sort

sắp tăng theo trật tự tự nhiên

`sort(v.begin(), v.end());`

sắp giảm theo trật tự tự nhiên

`sort(v.begin(), v.end(), greater<int>());`

sắp theo tiêu chí tự chọn với hàm bool Less

`sort(v.begin(), v.end(), Less);`

`swap(u,v)`

Hoán đổi trị của hai vector cùng kiểu u và v

Ví dụ

```
// swap
#include<bits/stdc++.h>
using namespace std;
typedef vector<int> VI;
void Print(VI v, const char * msg = "") {
    cout << msg;
    for(int x:v) cout << " " << x;
}
int main() {
    VI u, v;
    for (int i = 1; i < 10; ++i) {
        u.push_back(i);
        v.push_back(-i);
    }
    Print(u, "\n u: "); Print(v, "\n v: ");
    swap(u,v);
```

```

Print(u, "\n u: "); Print(v, "\n v: ");
cout << "\n T h e   E n d";
return 0;
}

```

Output

```

u:  1 2 3 4 5 6 7 8 9
v:  -1 -2 -3 -4 -5 -6 -7 -8 -9
u:  -1 -2 -3 -4 -5 -6 -7 -8 -9
v:  1 2 3 4 5 6 7 8 9
T h e   E n d

```

Minh họa

Chương trình sau đây sinh các hoán vị của 1, 2, 3

```

// Hoan vi 1,2,3
#include<bits/stdc++.h>
using namespace std;
typedef vector<int> VI;
typedef vector<VI> VVI;
typedef VVI::iterator IIVI;
void Print(VI v, const char * msg = "") {
    cout << msg;
    for(int i = 0; i < v.size(); ++i)
        cout << " " << v[i];
}
int main() {
    VVI vv;
    VI v;
    for(int a = 1; a <= 3; ++a) {
        for(int b = 1; b <= 3; ++b) {
            if (b != a) {
                int c = 6-a-b;
                v.clear();
                v.push_back(a); v.push_back(b ); v.push_back(c);
                vv.push_back(v);
                // hoan vi b, c
                v[1] = c; v[2] = b;
                vv.push_back(v);
                break;
            }
        }
    }
}

```

16 CHUYÊN TIN 11: CÁC CHUYÊN ĐỀ CHỌN LỌC VỚI NGÔN NGỮ C++

```
    } // if
    } // b
} // a
for(IVVI i = vv.begin(); i < vv.end(); ++i) {
    Print(*i, " ");
}
cout << "\n T h e   E n d";
return 0;
}
```

Output

```
1 2 3  1 3 2  2 1 3   2 3 1   3 1 2   3 2 1
T h e   E n d
```

Collatz Problem

Nhà toán học người Đức Lothar Collatz xây dựng dãy số mang tên ông là *dãy Collatz* hay $3n+1$ như sau:

Xuất phát từ số nguyên dương n .

Lặp đến khi $n = 1$ các thao tác sau:

Nếu n là số lẻ thì thay n bằng $3n + 1$.

Nếu n chẵn thì chia đôi n .

Ví dụ

Với $n = 3$ bạn sẽ nhận được dãy Collatz

gồm 8 số như sau:

3, 10, 5, 16, 8, 4, 2, 1.



Lothar Collatz
(1910-1990)

Ký hiệu

$C(n)$ là dãy Collatz sinh ra từ số tự nhiên $n \geq 1$, $\text{Len}(n)$ = chiều dài của dãy $C(n)$.

Ví dụ

$C(1) = [1]$, $\text{Len}(1) = 1$,

$C(3) = [3, 10, 5, 16, 8, 4, 2, 1]$, $\text{Len}(3) = 8$

Yêu cầu

a) Viết các hàm $C(n)$ và $\text{Len}(n)$

b) Số nào trong khoảng $1..1M$ (một triệu) có Len dài nhất ?

c) Với mỗi số n , hãy cho biết trong $C(n)$ có bao nhiêu số nguyên tố dưới $1M$?

Algorithm

a) Để ý rằng dù $n < 1M$ thì dãy Collatz có thể sinh ra những số lớn hơn 1M. Ví dụ, Bạn xuất phát từ $n = 999997 < 1000000$, do n lẻ nên bạn biến đổi:

$$n = 3n+1 = 3*999997 + 1 = 2999992 > 1000000$$

Do đó, nên dùng vector<long long> C chứa dãy số được sinh theo quy tắc Collatz (64 bit).

Program

```
// Collatz Problem a
#include<bits/stdc++.h>
using namespace std;
typedef unsigned long long UL;
typedef vector<UL> VL;
// Collatz sequence VL
VL C(UL n) {
    VL c;
    c.push_back(n);
    while(n > 1) {
        n = (n % 2) ? 3*n+1 : n/2;
        c.push_back(n);
    }
    return c;
}
int main() {
    VL c = C(3);
    cout << "\n a) ";
    for(UL x:c) cout << " " << x;
    cout << "\n Len = " << c.size();
    c = C(999997);
    cout << "\n b) ";
    for(UL x:c) cout << " " << x;
    cout << "\n Len = " << c.size();
    cout << "\n T h e   E n d";
    return 0;
}
```

Output

```
a)  3 10 5 16 8 4 2 1
Len = 8
```


18 CHUYÊN TIN 11: CÁC CHUYÊN ĐỀ CHỌN LỌC VỚI NGÔN NGỮ C++

b) 999997 2999992 1499996 749998 374999 1124998 562499 ...

Len = 114

T h e E n d

b) Ta dùng phương pháp quy hoạch động để tính $\max\{\text{Len}(n) \mid n = 1..1M\}$

Giả sử với n cho trước, ta đã tính được các $\text{Len}(k)$, $k = 1..n$. Ta cần tính $\text{Len}(n+1)$?

Ta sinh phần đầu dãy Collatz từ $n+1$ đến giá trị $m < n$. Gọi chiều dài của dãy này là h .

Khi đó $\text{Len}(n+1) = h + \text{Len}(m)$.

Bạn để ý rằng khi n là số lẻ thì $3n+1$ là số chẵn, do đó tại bước tiếp theo bạn sẽ có:

$$(3n + 1) \text{ div } 2 = (2n + n + 1) \text{ div } 2 = n + (n + 1) \text{ div } 2$$

do đó bạn có thể kết hợp hai bước này.

Program

```
// Collatz Problem b, answer 525 837799
#include<bits/stdc++.h>
using namespace std;
typedef unsigned long long UL;
const int MN = 1000000;
vector<int> len(MN+1);
void MaxLen(int limit) {
    len[0] = 0; len[1] = 1; len[2] = 2;
    for(int n = 3; n <= limit; ++n) {
        UL m = n;
        int h = 0;
        while(m >= n) {
            ++h;
            if(m % 2) { // m is odd
                // m = (3m+1)/2 = (2m+m+1)/2 = m + (m+1)/2
                m += (m+1)/2; ++h;
            } else m /= 2;
        }
        // m < n
        len[n] = len[m] + h;
    }
    int id = 0; // tìm max
    for(int i = 1; i <= limit; ++i)
        if (len[i] > len[id]) id = i;
    cout << "\n max Len = " << len[id] << " at " << id;
}
```

```
int main() {
    MaxLen(1000000); // 837799 Length 525
    cout << "\n T h e   E n d";
    return 0;
}
```

Output

```
max Len = 525 at 837799
T h e   E n d
```

c)

Algorithm

- ✂ Trước hết ta gọi thuật toán sàng để sinh ra các số nguyên tố không vượt quá 1M.
- ✂ Sau đó, ta dùng quy hoạch động để tính số lượng các số nguyên tố trong $C(n)$ giống như câu b.

Gọi $P(n)$ là số lượng các số nguyên tố dưới 1M trong $C(n)$, h là số lượng các số nguyên tố dưới 1M trong dãy Collatz $n..m$, $m < n$. Ta có hệ thức $P(n) = P(m) + h$.

Ta tính trước $P(0) = P(1) = 0$; $P(2) = 1$ (vì 2 nguyên tố). Đặt $mn = 2$ là giới hạn trong kho chứa (base) P .

Sau đó, với mỗi n ta xét:

- Nếu $n \leq m$ tức là $P(n)$ có sẵn trong kho ta chỉ việc xuất $P(n)$.
- Nếu $n > m$ tức là $P(n)$ không có sẵn trong kho, ta gọi thủ tục Extend để tính tiếp theo quy hoạch động các P , từ $P(mn+1)$ đến $P(n)$ rồi xuất $P(n)$.

Program

```
// Collatz Problem c
#include<bits/stdc++.h>
using namespace std;
typedef unsigned long long UL;
typedef vector<int> VI;
const int MN = 1000000;
bitset<MN> e; // prime sieve
VI p(MN+1); // primes < 1M in C(n)
int mn; // gioi han hien co
// cac so nguyen to < n
void Sieve(int n) {
    e.reset(); // e = 0..0
    for(int i = 2; i < n; ++i)
        e.set(i); // e[2..n-1] = 1
```

20 CHUYÊN TIN 11: CÁC CHUYÊN ĐỀ CHỌN LỌC VỚI NGÔN NGỮ C++

```
// gach cac boi cua 2 ke tu 4
for(int i = 4; i < n; i += 2) e[i] = 0;
int sn = (int)sqrt(n);
for(int i = 3; i <= sn; i += 2) { // xet cac so le
    if (e[i]) { // i nguyen to
        // gach cac boi cua i ke tu i*i
        for(int j = i*i; j < n; j += i) e[j] = 0;
    }
}
}

int Extend(int n) {
    while(mn < n) {
        ++mn;
        UL m = mn;
        int h = 0;
        while(m >= mn) {
            h += e[m];
            m = (m % 2) ? 3*m + 1 : m / 2;
        }
        // m < mn
        p[mn] = p[m] + h;
    }
    return p[n];
}

int main() {
    Sieve(MN);
    p[0] = 0; p[1] = 0; // 1 is not prime
    p[2] = 1; // 2 is prime
    mn = 2;
    // Tinh cac P(x), x:d
    int d[] = {13, 11, 7, 27, 10, 6, 200, 127, 100, 7};
    for(int n:d) {
        cout << "\n Number of primes below 1000000 in Collatz of "<< n;
        if (n <= mn) cout << "\n In base: " << p[n];
        else cout << "\n Extend to " << n
            << ": " << Extend(n);
    }
    cout << "\n T h e   E n d";
    return 0;
}
```

Output

```

Number of primes below 1000000 in Collatz of 13
Extend to 13: 3
Number of primes below 1000000 in Collatz of 11
In base: 5
Number of primes below 1000000 in Collatz of 7
In base: 6
Number of primes below 1000000 in Collatz of 27
Extend to 27: 25
Number of primes below 1000000 in Collatz of 10
In base: 2
Number of primes below 1000000 in Collatz of 6
In base: 3
Number of primes below 1000000 in Collatz of 200
Extend to 200: 7
Number of primes below 1000000 in Collatz of 127
In base: 12
. . .
T h e   E n d

```

Open Problem

Đến nay chưa có ai chứng minh hoặc bác bỏ giả thiết Collatz sau đây:

Phải chăng mọi dãy Collatz đều kết thúc với mọi n ?

Set

Set (tập hợp) trong C++ chứa các phần tử khác nhau.

Hầu hết các thao tác trên set giống với các thao tác trên vector.

```

typedef set<T> ST; // khai báo set kiểu T
typedef ST::iterator IST; // khai báo kiểu iterator
ST s; // khai báo biến s
s.insert(e); // nạp e vào set s, loại trùng lặp
s.erase(e); // xóa e khỏi set s nếu e có trong s, không có thì bỏ qua
s.clear(); // xóa hết, giữ lại biến s
bool s.count(e); // e có trong s ? count = 1|0
s.size();
bool s.empty(); // kiểm tra s rỗng ?
for(auto x:s) F(x);

```

Symmetric Points

Cho n điểm nguyên phân biệt trên trục số và m điểm nguyên làm tâm đối xứng $c_1, c_2, \dots, c_m$. Hãy cho biết với mỗi tâm đối xứng c_i có bao nhiêu cặp trong số n điểm đã cho đối xứng qua tâm đó?

Các tâm đối xứng không trùng nhau và không trùng với bất kỳ điểm nào trong số n điểm đã cho.

Các điểm và tâm đối xứng nằm trong khoảng $-1M$ đến $1M$.

POINT . INP	OUTPUT
10 3	3
5 2 -1 -2 7 3 -5 1 9 8	0
4 -4 6	2

Input cho biết có 10 điểm trên trục số và 3 tâm đối xứng là 4, -4 và 6.

Output cho biết

✧ có 3 cặp điểm đối xứng qua tâm 4 ((3, 5), (-1, 9), (1, 7)).

✧ không có cặp điểm nào đối xứng qua tâm -4

✧ có 2 cặp điểm đối xứng qua tâm 6 ((5, 7), (3, 9)).

-5	-4	-3	-2	-1	0	1	2	3	4	5	6	7	8	9
x			x	c		b	x	a	*	a		b	x	c

Tâm đối xứng 4: (-1, 9), (1, 7), (3, 5)

-5	-4	-3	-2	-1	0	1	2	3	4	5	6	7	8	9
x	*		x	x		x	x	x		x		x	x	x

Tâm đối xứng -4:

-5	-4	-3	-2	-1	0	1	2	3	4	5	6	7	8	9
x			x	x		x	x	b		a	*	a	x	b

Tâm đối xứng 6: (5, 7), (3, 9)

Algorithm

Với mỗi tâm đối xứng c ta lưu khoảng cách từ mỗi điểm x đến tâm c , $\text{abs}(x-c)$ vào set d . Vì d không chứa các giá trị trùng lặp nên hiệu $n-d.\text{size}$ sẽ chính là số lượng các cặp điểm đối xứng qua tâm c .

Độ phức tạp: $O(mn)$

Program

```
// Symmetric Points
#include<bits/stdc++.h>
using namespace std;
```

```

const char * FN = "POINT.INP";
typedef vector<int> VI;
typedef set<int> SI;
void Run() {
    ifstream f(FN);
    if (!f) {
        cout << "\n unable open input file " << FN;
        exit(0);
    }
    int n, m;
    f >> n >> m;
    cout << "\n number of points: " << n;
    VI p(n); // points
    for(int i = 0; i < n; ++i) f >> p[i];
    cout << "\n number of centers: " << m;
    VI c(m);
    for(int i = 0; i < m; ++i) f >> c[i];
    f.close();
    cout << "\n Points: ";
    for(auto x:p) cout << " " << x;
    SI d; // set of int
    for(auto t:c) {
        cout << "\n Center: " << t;
        d.clear(); // voi moi center c can xoa d
        for(auto x:p) d.insert(abs(x-t));
        cout << "\n has: " << n-d.size()
            << " symmetric pairs";
    }
}
int main() {
    Run();
    cout << "\n T h e   E n d";
    return 0;
}

```

Output

```

number of points: 10
number of centers: 3
Points:  -5 -2 -1 1 2 3 5 7 9 8

```

24 CHUYÊN TIN 11: CÁC CHUYÊN ĐỀ CHỌN LỌC VỚI NGÔN NGỮ C++

```
Center: 4
has: 3 symmetric pairs
Center: -4
has: 0 symmetric pairs
Center: 6
has: 2 symmetric pairs
T h e   E n d
```

Cách sinh test

- ☞ Đánh dấu ngẫu nhiên một số điểm trong mảng $a[0..m]$ làm dữ liệu vào cho dãy số.
 - ☞ Chọn một số điểm chưa đánh dấu làm tâm đối xứng.
 - ☞ Ghi dữ liệu lùi lại một khoảng $d < m$, ví dụ $d = 3$
- Dãy số: -3, 0, 3, 4, 5, 7, 9, 10, 11
Tâm đối xứng: -1, 1, 6

Program with Auto Run

```
// Symmetric Points
#include<bits/stdc++.h>
using namespace std;
typedef vector<int> VI;
typedef set<int> SI;
VI p; // points
VI c; // center
void Exe() {
    cout << "\n Points: ";
    for(auto x:p) cout << " " << x;
    SI d; // set of int
    for(auto t:c) {
        cout << "\n Center: " << t;
        d.clear(); // voi moi center c can xoa d
        for(auto x:p) d.insert(abs(x-t));
        cout << "\n has: " << p.size()-d.size()
            << " symmetric pairs";
    }
}
// k center
void AutoRun(int n, int k) {
    cout << "\n Auto Run: ";
```

```

int nn = n*3;
int a[nn];
memset(a, 0, nn*sizeof(int)); // sat a = 0...
srand(time(NULL)); // khoi dong bo so ngau nhien
// chon ngau nhien cac so cho p
for(int i = 0; i < n; ++i) a[rand() % nn] = 1;
// chon ngau nhien k tam doi xung cho c
int j;
for(int i = 0; i < k; ++i) {
    while(true) {
        j = rand() % nn;
        if(a[j] == 0) break;
    }
    a[j] = 2;
}
cout << "\n a: ";
for(auto x:a) cout << " " <<x;
// ghi du lieu
p.clear(); c.clear();
int d = n/3; // khoang lui
for(int i = 0; i < nn; ++i) {
    if(a[i] == 1) p.push_back(i-d);
    else if(a[i] == 2) c.push_back(i-d);
}
Exe();
}
int main() {
    AutoRun(20, 4);
    cout << "\n T h e   E n d";
    return 0;
}

```

Output

Auto Run:

a: 0 0 1 0 0 1 0 0 0 1 2 0 1 1 0 0 2 0 0 0 0 0 0 1 0 0 1 0 0 1 1 0 0

1 1 1 2 1 0 2 0 1 0 0 0 0 0 0 0 1 0 0 1 0 0 1 0 1 0 0 0

Points: -4 -2 2 5 6 16 19 22 23 26 27 28 30 34 42 45 48 50

Center: 3

has: 0 symmetric pairs

26 CHUYÊN TIN 11: CÁC CHUYÊN ĐỀ CHỌN LỌC VỚI NGÔN NGỮ C++

```
Center: 9
has: 2 symmetric pairs
Center: 29
has: 2 symmetric pairs
Center: 32
has: 4 symmetric pairs
T h e   E n d
```

List

List (danh sách) là kiểu dữ liệu gồm dãy các phần tử cùng kiểu, truy nhập tuần tự (tuyến tính), tức là không chấp nhận kí pháp `a[i]` hoặc `a.at(i)`.

List hoạt động giống vector ngoại trừ khả năng nạp trị vào cả hai đầu và không truy nhập trực tiếp.

```
Khai báo kiểu list kiểu T: typedef list<T> LT;
Khai báo iterator: LT::iterator ILT;
Khai báo biến a kiểu LT: LT a;
Nạp trị x vào cuối list a: a.push_back(x);
Nạp trị x vào đầu list a: a.push_front(x);
Loại bỏ phần tử đầu list: a.pop_front();
Loại bỏ phần tử cuối list: a.pop_back();
Thực hiện thao tác F(x) trên các phần tử của list a:
for(T x:a) F(x);
or
for(auto x:a) F(x);
Tự lật list a: a.reverse();
Delete all element = x: a.remove(x);
Xóa các phần tử thỏa điều kiện s trong list a
Ví dụ, xóa các số chẵn
bool IsEven(long n) { return (n % 2 == 0); }
a.remove_if(IsEven); // delete all even numbers in a
sort: a.sort();
unique(): Nếu list a được sắp thì a.unique() loại bỏ các phần tử trùng lặp
trong a.
count: return 1 (true) nếu x có trong a; ngược lại return 0 (false).
bool count(a.begin(), a.end(), x);
```

Demo

Chương trình dưới đây demo các thao tác chủ yếu trên int list.

```
// List demo
#include<bits/stdc++.h>
```

```

using namespace std;
typedef list<int> LI;
bool Is_Negative(int x) { return x < 0; }
void Demo() {
    LI a; // khai bao list a empty
    cout << "\n is a empty ? " << a.empty();
    // Insert 1..5 to the end of a
    for(int i = 1; i <= 5; ++i) a.push_back(i);
    cout << "\n Insert 1..5 to the end of a: ";
    for(auto x:a) cout << " " << x;
    // Insert -11..-15 to the head of a
    for(int i = 1; i <= 5; ++i) a.push_front(-(i+10));
    cout << "\n Insert -11..-15 to the head of a: ";
    for(auto x:a) cout << " " << x;
    // delete elements at head and at end of a
    a.pop_front(); a.pop_back();
    cout << "\n delete elements at head and at end of a: ";
    for(auto x:a) cout << " " << x;
    a.reverse();
    cout << "\n reversed a: ";
    for(auto x:a) cout << " " << x;
    for(auto x:a) {
        cout << "\n Is " << x << " in a ? "
            << count(a.begin(), a.end(), x);
        int y = -x;
        cout << "\n Is " << y << " in a ? "
            << count(a.begin(), a.end(), y);
    }
    a.sort();
    cout << "\n sorted a: ";
    for(auto x:a) cout << " " << x;
    // delete all negative elements
    a.remove_if(Is_Negative);
    cout << "\n delete all negative elements: ";
    for(auto x:a) cout << " " << x;
    LI b(a);
    cout << "\n new list b = a: ";
}

```

28 CHUYÊN TIN 11: CÁC CHUYÊN ĐỀ CHỌN LỌC VỚI NGÔN NGỮ C++

```
    for(auto x:b) cout << " " << x;
    // b = b + a
    for(auto x:a) b.push_back(x);
    cout << "\n b = b + a: ";
    for(auto x:b) cout << " " << x;
    b.unique();
    cout << "\n unique b if not sort: ";
    for(auto x:b) cout << " " << x;
    b.sort();
    cout << "\n sorted b: ";
    for(auto x:b) cout << " " << x;
    b.unique();
    cout << "\n unique sorted b: ";
    for(auto x:b) cout << " " << x;
}
int main() {
    Demo();
    cout << "\n T h e   E n d";
    return 0;
}
```

Output

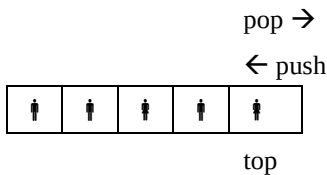
```
is a empty ? 1
Insert 1..5 to the end of a:  1 2 3 4 5
Insert -11..-15 to the head  of a:  -15 -14 -13 -12 -11 1 2 3 4 5
delete elements at head and at end of a:  -14 -13 -12 -11 1 2 3 4
reversed a:  4 3 2 1 -11 -12 -13 -14
Is 4 in a ? 1
Is -4 in a ? 0
Is 3 in a ? 1
Is -3 in a ? 0
Is 2 in a ? 1
. . .
T h e   E n d
```

Stack

Ngăn xếp (stack) là kiểu dữ liệu được tổ chức theo nguyên tắc *vào sau - ra trước* (*LIFO: Last In First out*): phần tử được nạp vào sau thì sẽ được lấy ra trước. Khi ta tổ chức các hoạt động sau đây sẽ thường tuân thủ theo nguyên tắc của stack (st):

- ✧ Tháo các chi tiết của một sản phẩm để sửa chữa, sau đó lắp lại các chi tiết
- ✧ Xếp nguyên liệu vào các thùng đựng dạng đường ống với một đầu vào, sau đó lấy ra để sử dụng

Có thể tưởng tượng ngăn xếp như một đường ống st chỉ có một cửa đảm nhận chức năng vào/ra. Khi nạp một đối tượng x vào st, st sẽ được tăng thêm một ngăn (trên cùng) để chứa x, khi cần lấy khỏi st một phần tử thì st sẽ xuất ra phần tử trên cùng và do đó, số lượng sẽ được giảm đi 1.



Các thao tác cơ bản trên stack

stack có hai thao tác chính là nhập (push) và xuất (pop) dữ liệu.

Hiển thị kèm chú thích

```
void Print(SI s, const char * msg = "") {
    cout << msg;
    if (s.empty()) return;
    int x = s.top(); // get
    s.pop(); // delete
    Print(s); // recursive
    cout << " " << x;
}
```

Ghi nhớ

```
typedef stack<T> ST; // khai báo kiểu
ST s; // khai báo biến
s.empty(); // kiểm tra tính rỗng
s.size(); // số phần tử
s.push(x); // nạp x vào stack s
s.top(); // xem ngăn trên của s
s.pop(); // xuất ngăn trên của s, giảm số lượng 1 đơn vị
```

Demo stack

Chương trình dưới đây demo các thao tác cơ bản trên stack int.

```
// stack demo
#include<bits/stdc++.h>
using namespace std;
typedef stack<int> STI;
```

30 CHUYÊN TIN 11: CÁC CHUYÊN ĐỀ CHỌN LỌC VỚI NGÔN NGỮ C++

```
void Print(STI s, const char * msg = "") {
    cout << msg;
    if (s.empty()) return;
    int x = s.top();
    s.pop();
    Print(s); // recursive
    cout << " " << x;
}

void Demo() {
    STI s; // khai bao stack s
    Print(s, "\n s: ");
    cout << "\n is s empty ? " << s.empty();
    cout << "\n size = " << s.size();
    // Insert 1..5 to the top (end) of s
    for(int i = 1; i <= 5; ++i) s.push(i);
    Print(s, "\n Insert 1..5: ");
    cout << "\n is s empty ? " << s.empty();
    cout << "\n size = " << s.size();
    for(int i = 1; i <= 5; ++i) s.pop();
    Print(s, "\n pop all data from s: ");
    cout << "\n is s empty ? " << s.empty();
    cout << "\n size = " << s.size();
}

int main() {
    Demo();
    cout << "\n T h e   E n d";
    return 0;
}
```

Output

```
s:
is s empty ? 1
size = 0
Insert 1..5:  1 2 3 4 5
is s empty ? 0
size = 5
pop all data from s:
is s empty ? 1
size = 0
T h e   E n d
```

Cài đặt stack qua vector

Trong C++, ngắn xếp được cài đặt qua kiểu dữ liệu `vector<T>` như sau:

```
Tyedef stack kiểu T: typedef vector<T> VT;
Khởi tạo stack st kiểu T: VT st;
Dung lượng: len = st.size();
st.clear(); // Xóa hết nội dung st; giữ lại biến st
st.back(); // xem ngọn st
st.empty(); // Kiểm tra st rỗng ? 1: rỗng, 0: không rỗng
st.push_back(x); // nạp trị x vào ngọn st, dung lượng của stack tăng 1 đơn vị
st.pop_back(); // hoặc q.erase(q.end()-1): Lấy đối tượng trên ngọn st ra khỏi
st, dung lượng st giảm bớt 1 đơn vị.
Hiển thị st kèm thông báo msg:
void Print(VI s, const char * msg = "") {
    cout << msg;
    for(int x:s) cout << " " << x;
}
```

Demo stack using vector

```
// stack demo using vector
#include<bits/stdc++.h>
using namespace std;
typedef vector<int> VI;
void Print(VI s, const char * msg = "") {
    cout << msg;
    for(int x:s) cout << " " << x;
}
void Demo() {
    VI s; // khai bao stack s
    Print(s, "\n s: ");
    cout << "\n is s empty ? " << s.empty();
    cout << "\n size = " << s.size();
    // Insert 1..5 to the top (end) of s
    for(int i = 1; i <= 5; ++i) s.push_back(i);
    Print(s, "\n Insert 1..5: ");
    cout << "\n is s empty ? " << s.empty();
    cout << "\n size = " << s.size();
    for(int i = 1; i <= 5; ++i) s.pop_back();
    Print(s, "\n pop all data from s: ");
}
```

32 CHUYÊN TIN 11: CÁC CHUYÊN ĐỀ CHỌN LỌC VỚI NGÔN NGỮ C++

```
    cout << "\n is s empty ? " << s.empty();
    cout << "\n size = " << s.size();
}
int main() {
    Demo();
    cout << "\n T h e   E n d";
    return 0;
}
```

Output

```
s:
is s empty ? 1
size = 0
Insert 1..5:  1 2 3 4 5
is s empty ? 0
size = 5
pop all data from s:
is s empty ? 1
size = 0
T h e   E n d
```

Cài đặt stack qua array

Trong C++, nếu biết trước kích thước tối đa n bạn có thể cài đặt nhanh ngăn xếp qua array như sau:

```
int s[n]; // khai báo ngăn xếp s
int p = -1; // trỏ vào ngăn trên của s
p < 0 // empty ?
p+1; // size
s[++p] = x; // push
--p; // pop
```

Demo stack using array

```
// stack demo using array
#include<bits/stdc++.h>
using namespace std;
void Print(int s[], int p, const char * msg = "") {
    cout << msg;
    for(int i = 0; i <= p; ++i)
        cout << " " << s[i];
}
```

```

void Demo() {
    int s[100];
    int p = -1; // pointer to the top
    Print(s, p, "\n s: ");
    cout << "\n is s empty ? " << (p < 0);
    cout << "\n size = " << p+1;
    // Insert 1..5 to the top (end) of s
    for(int i = 1; i <= 5; ++i) s[++p] = i;
    Print(s, p, "\n Insert 1..5: ");
    cout << "\n is s empty ? " << (p < 0);
    cout << "\n size = " << p+1;
    for(int i = 1; i <= 5; ++i) --p;
    Print(s, p, "\n pop all data from s: ");
    cout << "\n is s empty ? " << (p < 0);
    cout << "\n size = " << p+1;
}

int main() {
    Demo();
    cout << "\n T h e   E n d";
    return 0;
}

```

Output

```

s:
is s empty ? 1
size = 0
Insert 1..5:  1 2 3 4 5
is s empty ? 0
size = 5
pop all data from s:
is s empty ? 1
size = 0
T h e   E n d

```

Brackets

Một biểu thức toán học có thể chứa các cặp ngoặc thuộc ba dạng (), [] hoặc { }.

Hãy kiểm tra tính hợp lệ của các biểu thức có chứa các cặp ngoặc?

34 CHUYÊN TIN 11: CÁC CHUYÊN ĐỀ CHỌN LỌC VỚI NGÔN NGỮ C++

Ví dụ

BRACKET.INP	OUTPUT
a*(b+c*d)	Correct
a*(b+c*d))	Syntax Error: Unmatched)
a*([b+c]*d)	Correct
{a*[b+c]-d}	Correct
{a*{[b+c]*d)}	Syntax Error: Unmatched)
{a*{(b+c)*d}}	Syntax Error: Unmatched]
{a*{(b+c)*d}}	Correct
({a*{(b+c)*d}}	Syntax Error: Unmatched (

Understanding

Ta cần hiểu thế nào là một cặp ngoặc được bố trí hợp lệ?

Đời thường có thể hiểu mỗi cặp ngoặc gồm có nôi (dấu mở ngoặc) và vung (dấu đóng ngoặc tương ứng). Như vậy, các cặp ngoặc cùng loại có thể chứa nhau giống như một dãy nôi-vung có thể đặt lồng nhau.

Algorithm

Ta thể hiện dãy nôi-vung dưới dạng một stack st. Gặp dấu mở ngoặc (nôi) ta nạp vào st để chờ sau này sẽ gặp vung tương ứng. Gặp dấu đóng ngoặc (vung) ta xét ngọn st có chứa nôi cùng loại hay không. Nếu có thì ta lấy nôi đó ra để thu được một cặp nôi-vung cùng loại. Để thấy hai tình huống không hợp lệ khi gặp vung (đóng ngoặc) sau đây:



✂ st rỗng: trên ngăn chờ không có gì

✂ trên ngăn chờ không có nôi cùng loại.

Cuối cùng, bạn kiểm tra lại stack. Nếu st không rỗng, tức là còn có những dấu mở ngoặc (nôi) đang chờ trên st mà chưa gặp các dấu đóng ngoặc (vung) tương ứng thì biểu thức có lỗi.

Để tránh việc xử lý rỗng ta đặt một phần tử đệm # vào stack, như vậy lời gọi st.top() luôn có hiệu lực.

Program

```
// Brackets
#include<bits/stdc++.h>
using namespace std;
const char * FN = "BRACKET.INP";
```

```

inline void Go() {
    cout << " ? ";
    fflush(stdin);
    if (cin.get() == '.') exit(0);
}

inline void Err(char c) {
    cout << "Syntax Error: Unmatched " << c;
}

inline GetOpen(char c) {
    switch(c) {
        case ')': return '(';
        case ']': return '[';
        case '}': return '{';
    }
}

// remove all element != c in s
string Trim(string s, char c = 32) {
    string w = "";
    for (auto e:s)
        if (e != c) w += e;
    return w;
}

void Test(string s) {
    cout << "\n Test " << s << " : ";
    stack<char> st;
    st.push('#'); // guard
    for (auto c:s) {
        switch(c) {
            // gap mo ngoac
            case '(':
            case '[':
            case '{': st.push(c); break;
            // gap dong ngoac
            case ')':
            case ']':
            case '}': if(st.top() != GetOpen(c)) {

```

36 CHUYÊN TIN 11: CÁC CHUYÊN ĐỀ CHỌN LỌC VỚI NGÔN NGỮ C++

```
        Err(c); return;
    } else st.pop();
    break;

    } // switch
}
if (st.top() == '#') cout << "Correct.";
else Err(st.top());
return;
}

void Run() {
    ifstream f(FN);
    if (!f) {
        cout << "\n unable open input file " << FN;
        exit(0);
    }
    string s = "";
    while (getline (f,s) ) {
        s = Trim(s); // bo BL dau va cuoi s
        if (s == "") break;
        Test(s);
    }
}

int main() {
    Run();
    cout << "\n T h e   E n d";
    return 0;
}
```

Output

```
Test a*(b+c*d) : Correct.
Test a*(b+c*d)) : Syntax Error: Unmatched )
Test a*([b+c]*d) : Correct.
Test {a*[b+c]-d} : Correct.
Test {a*{[b+c]*d)) : Syntax Error: Unmatched )
Test {a*{(b+c)*d}} : Syntax Error: Unmatched ]
Test {a*{(b+c)*d}} : Correct.
Test ({a*{(b+c)*d}} : Syntax Error: Unmatched (
T h e   E n d
```

Cách sinh Test

- ✧ Viết các biểu thức đúng
- ✧ sửa lại một vài vị trí thành sai

Postfix Syntax

Năm 1924, nhà toán học người Ba Lan - Jan Poland Lukasiewicz đề xuất phương pháp ghi các biểu thức toán học dưới dạng *dấu phép toán được đặt sau các toán hạng*, ví dụ, thay vì viết $c \times (d + h)$ thì ta viết $c \ d \ h \ + \ \times$

Ký pháp này có đặc điểm là không phải dùng đến các dấu ngoặc để chỉ rõ thứ tự thực hiện các phép tính trung gian. Sau khoảng gần nửa thế kỷ, khi máy tính điện tử ra đời (năm 1949), các nhà tin học phát hiện ra rằng nếu chuyển các biểu thức toán học sang dạng hậu tố thì rất tiện lợi cho việc thực hiện tính toán kết quả trên một ngăn xếp. Có lẽ, vì thế mà mọi người thường gọi ký pháp hậu tố là ký pháp Ba Lan nhằm tôn vinh quê hương ông.



Jan Lukasiewicz
(1878-1956)

pháp

Tính giá trị của biểu thức hậu tố

Gia sử ta có một biểu thức hậu tố dưới dạng string

`pol = "c d h + * "`

Đây là dạng Poland của biểu thức chính quy $c * (d + h)$. Ta sẽ triển khai hàm `Exe(pol)` tính trị của biểu thức `pol` với các quy ước sau đây:

- ✧ Mỗi biến trong `pol` có tên là một chữ cái viết thường trong bảng chữ cái tiếng Anh, `a, b, c, ...` và được nhận giá trị mặc định theo thứ tự $a = 0, b = 1, \dots, z = 25$.

Hàm `Exe` khi đó sẽ được thực hiện trên một ngăn xếp `st` kiểu `int` như sau:

`Exe(pol):`

 Khởi tạo ngăn xếp rỗng `st`

 for each symbol `c` in `pol` do

 if `c` là biến: nạp trị của `c` vào `st`

 else if `c` là phép toán `@` :

 thực hiện phép toán `@` trên các toán hạng nằm trên ngọn của `st`,

 đồng thời lấy các toán hạng ra khỏi ngọn `st`,

 ghi kết quả trung gian vào ngọn của `st`

 end if

 return: kết quả là giá trị trên ngọn của `st`

38 CHUYÊN TIN 11: CÁC CHUYÊN ĐỀ CHỌN LỌC VỚI NGÔN NGỮ C++

Ta minh họa quá trình tính trị của dạng hậu tố Poland pol = "c d h + *" như sau:

Exe("c d h + *") = ?		
ký tự		st
	Khởi trị	[]
c	nạp Val(c) = 2 vào st:	[2]
d	nạp Val(d) = 3 vào st:	[2, 3]
h	nạp Val(h) = 7 vào st:	[2, 3, 7]
+	thực hiện st[1] += st[2]:	[2, 10]
*	thực hiện st[0] *= st[1]:	[20]
	return	20

Cài đặt stack đơn giản qua một mảng

Nếu biết trước độ lớn của stack, bạn có thể sử dụng một mảng st để cài đặt stack như sau:

```
Khởi tạo stack với dung lượng tối đa MN: int a[MN] st;
Khởi tạo con trỏ dưới đáy stack: p = -1; (đáy p = 0)
Nạp trị v vào st: st[++p] = v;
Xem ngọn st: st[p];
Loại bỏ ngọn st: --p;
Kiểm tra st rỗng: p < 0;
```

Program

```
// Postfix expression
#include<bits/stdc++.h>
using namespace std;
// Print array a[d..c] with message msg
void Print(int a[], int d, int c, const char* msg = "") {
    cout << msg;
    for (int i = d; i <= c; ++i) cout << a[i] << " ";
}
// c is variable
inline bool IsVar(char c) { return (c >= 'a') && (c <= 'z'); }
// get value of var c
inline int Val(char c) { return c - 'a'; }
void Err() {
    cout << "\n Run error: divide by zero.";
    exit(0);
}
```

```

void Exe(string pol) {
    cout << "\n Input Postfix expression: " << pol;
    int st[pol.length()]; // int stack
    int p = -1; // index of top stack
    for(int i = 0; i < pol.length(); ++i) {
        char c = pol[i];
        if (IsVar(c)) {st[++p] = Val(c); continue;}
        switch(c) { // c is operator +-*/%
            case '+': st[p-1] += st[p]; --p; break;
            case '-': st[p-1] -= st[p]; --p; break;
            case '*': st[p-1] *= st[p]; --p; break;
            case '/': if (st[p] == 0) Err();
                    st[p-1] /= st[p]; --p; break;
            case '%': if (st[p] == 0) Err();
                    st[p-1] %= st[p]; --p; break;
        } // switch
    } // for
    cout << "\n " << st[p];
}

int main() {
    Exe("c d h + *"); // 20
    cout << "\n T h e   E n d";
    return 0;
}

```

Output

```

Input Postfix expression: c d h + *
20
T h e   E n d

```

Testing

Bạn hãy mở rộng chương trình trên để có thể thực hiện các phép toán sau đây:

- Các phép toán 2 ngôi +, -, *, /, %
- Các phép toán 1 ngôi:
 - $\sim x$ đảo dấu của x : cho ra giá trị $-x$
 - $x >$ tăng x thêm 1: cho ra giá trị $x+1$
 - $x <$ giảm x bớt 1: cho ra giá trị $x-1$
 - $x!$ tính giai thừa $x! = 1*2*... *n$

40 CHUYÊN TIN 11: CÁC CHUYÊN ĐỀ CHỌN LỌC VỚI NGÔN NGỮ C++

Hãy chuyển đổi các biểu thức sau đây sang dạng Pol và test chương trình để kiểm tra kết quả:

```
c+d*k
d*d - c*d*h + h*h
c-d>!+k*d
(k-c*d)-(c+(d+b*e))
-(e-b)
-(-e-b)
```

Đáp án

```
c+d*k -> cdk*+ : 32
d*d-c*d*h+h*h -> dd*cd*h*-hh*+ : 16
c-d>!+k*d -> cd>!--kd*+ : 8
(k-c*d)-(c+(d+b*e)) -> kcd*-cdbe*++- : -5
-(e-b) -> eb-~ : -3
-(-e-b) -> e~b-~ : 5
```

Program

```
// Test for Postfix expression
#include<bits/stdc++.h>
using namespace std;
// Print array a[d..c] with message msg
void Print(int a[], int d, int c, const char* msg = "") {
    cout << msg;
    for (int i = d; i <= c; ++i)
        cout << a[i] << " ";
}
// c is variable
inline bool IsVar(char c) { return (c >= 'a') && (c <= 'z'); }
// get value of var c
inline int Val(char c) { return c - 'a'; }
void Err(int e, int v) {
    cout << "\n Run error ";
    switch(e) {
        case 1: cout << " divide by zero: " << v; break;
        case 2: cout << " negative parameter: " << v; break;
    }
    exit(0);
}
```

```

int Factorial(int k) {
    int v = 1;
    for(int i = 2; i <= k; ++i) v *= i;
    return v;
}

void Exe(string pol) {
    cout << "\n Input Postfix expression: " << pol;
    int st[pol.length()]; // int stack
    int p = -1; // index of top stack
    for(auto c:pol) {
        if (IsVar(c)) {st[++p] = Val(c); continue;}
        switch(c) { // c is operator +-*/%
            case '~': st[p] = -st[p]; break;
            case '>': ++st[p]; break;
            case '<': --st[p]; break;
            case '!': if (st[p] < 0) Err(2, st[p]);
                    st[p] = Factorial(st[p]); break;
            case '+': st[p-1] += st[p]; --p; break;
            case '-': st[p-1] -= st[p]; --p; break;
            case '*': st[p-1] *= st[p]; --p; break;
            case '/': if (st[p] == 0) Err(1, st[p]);
                    st[p-1] /= st[p]; --p; break;
            case '%': if (st[p] == 0) Err(1, st[p]);
                    st[p-1] %= st[p]; --p; break;
        } // switch
    } // for
    cout << " : " << st[p];
}

int main() {
    Exe("c d k * + "); // c+d*k -> cdk*+ : 32
    // d*d-c*d*h+h*h -> dd*cd*h*-hh*+ : 16
    Exe("dd*cd*h*-hh*+ "); // 16
    Exe(" cd>!--kd*+ "); // c-d>!--k*d -> cd>!--kd*+ : 8
    Exe(" kcd*-cdbe*++-"); // (k-c*d)-(c+(d+b*e)) -> kcd*-cdbe*++- : -5
    Exe(" eb~"); // -(e-b) -> eb~ : -3
    Exe(" e~b~"); // -(-e-b) -> e~b~ : 5
    cout << "\n T h e   E n d";
    return 0;
}

```


42 CHUYÊN TIN 11: CÁC CHUYÊN ĐỀ CHỌN LỌC VỚI NGÔN NGỮ C++

Output

```
Input Postfix expression: c d k * + : 32
Input Postfix expression: dd*cd*h*-hh*+ : 16
Input Postfix expression: cd>!--kd*+ : 8
Input Postfix expression: kcd*-cdbe*++- : -5
Input Postfix expression: eb-~ : -3
Input Postfix expression: e~b-~ : 5
T h e    E n d
```

Chuyển đổi biểu thức chính quy sang dạng hậu tố

Để chuyển biểu thức chính quy e sang dạng hậu tố chúng ta cần những quy tắc sau đây:

Quy tắc xây dựng biểu thức chính quy

Các biểu thức được xây dựng từ các biến a, b, c,... và các phép toán như đã nêu trên.

Quy định thứ tự ưu tiên của các phép toán

Khi tính toán giá trị một biểu thức số học, người ta thường quy định một trật tự ưu tiên như sau:

1. Tính trị của các biểu thức trong ngoặc
2. Tính trị các lời gọi hàm
3. Tính trị các phép toán một ngôi
4. Tính trị các phép toán hai ngôi

Phép toán có độ ưu tiên nhỏ thì được ưu tiên hơn.

Nếu hai phép toán có cùng độ ưu tiên thì thực hiện từ trái qua phải.

Ví dụ, biểu thức chính quy

$$e = c - d > ! + k * d$$

sẽ được thực hiện theo trật tự sau đây:

```
t1 = d>
t2 = t1!
t3 = c-t2
t4 = k*d
t5 = t3+t4
```

Như vậy dạng hậu tố chính là một biểu thức thể hiện *trật tự tính toán* nói trên:

$$pol = c \ d \ > \ ! \ - \ k \ d \ * \ +$$

Chú thích

Một số hệ thống tính toán quy định dãy các phép toán lũy thừa được tính từ phải qua trái, ví dụ, $2^2^2^3 = 2^{(2^3)} = 2^8 = 256$.

Chúng ta tạm quy định thứ tự ưu tiên cho các phép toán như sau:

Phép toán	Độ ưu tiên
>	100
<	100
!	100
*	200
/	200
%	200
+	300
-	300

Độ ưu tiên cách nhau xa để phòng khi có thêm phép toán mới thì vẫn có đủ chỗ cấp phát.

Algorithm

Gọi $\text{Compile}(e)$ là hàm chuyển đổi biểu thức chính quy e sang dạng Poland pol.

Ta duyệt từng ký tự c trong e và xét các tình huống:

- ✧ Nếu c là biến: đưa c ra pol
- ✧ Nếu c là phép toán $@$: chuyển ra pol các phép toán đã nạp trước trên stack có độ ưu tiên bằng hoặc nhỏ hơn độ ưu tiên của $@$. Sau đó nạp phép toán $@$ vào stack để chờ xử lý sau.
- ✧ Nếu c là dấu mở ngoặc '(': đưa c lên stack để chờ sau này gặp ')' sẽ xử lý.
- ✧ Nếu c là dấu đóng ngoặc ')': chuyển ra pol các phép toán trên ngọn stack cho đến khi gặp dấu mở ngoặc '(' thì bỏ đi dấu này khỏi stack.

Cuối cùng ta vét nốt các phép toán trên stack đưa ra pol.

Mình họa

Compile("c*(d + e - b*h)") = ?			
		st	pol
	Khởi trị	[]	empty
c	c → pol		c
*	* → st	[*]	
(	(→ st	[* , (]	
d	d → pol		cd
+	+ → st	[* , (, +]	
e	e → pol		cde
-	+ → pol - → st	[* , (, -]	cde+

44 CHUYÊN TIN 11: CÁC CHUYÊN ĐỀ CHỌN LỌC VỚI NGÔN NGỮ C++

Compile("c*(d + e - b*h)") = ?			
b	b → pol		cde+b
*	* → st	[, (, -, *]	
h	h → pol		cde+bh
)	st → pol đến khi gặp (thì bỏ (	[*]	cde+bh*-
	Hết input: all st → pol		cde+bh*-*
	return pol		cde+bh*-*

Một phép toán một ngôi có thể được sinh ra trong qua trình chuyển đổi biểu thức sang dạng hậu tố là phép đổi dấu. Ví dụ $e = -b$ cần được hiểu là đảo dấu của b , do đó $-b$ cần được dịch sang dạng hậu tố là $b\sim$, với $\sim$ là ký hiệu phép toán đảo dấu. Quan sát kỹ bạn sẽ phát hiện ra rằng phép toán $-$ được chuyển thành phép đảo dấu khi và chỉ khi nó là phép toán 1 ngôi, cụ thể là khi $-$ đứng đầu biểu thức hoặc ngay sau dấu '('. Ví dụ:

$-(b+c)$ sẽ được dịch thành $bc+\sim$

$(-b+c)$ sẽ được dịch thành $b\sim c+$

Để tiện xử lý, ta bỏ các dấu cách trong biểu thức e và thêm phần tử đệm '#' vào đầu biểu thức e và ngăn đầu tiên (ngăn số 0) của stack.

Chương trình dưới đây thực hiện hai pha: Pha đầu tiên dịch biểu thức toán học e sang dạng hậu tố pol. Pha thứ hai thực hiện tính toán trên pol. Cả hai pha đều sử dụng stack.

Program

```
// Expression
#include<bits/stdc++.h>
using namespace std;
const char SHAP = '#';
const char BL = 32;
const string OP = "~!><+-*/%";
// Print array a[d..c] with message msg
void Print(int a[], int d, int c, const char* msg = "") {
    cout << msg;
    for (int i = d; i <= c; ++i) cout << a[i] << " ";
}
inline bool IsVar(char c) { return (c >= 'a') && (c <= 'z'); }
inline bool IsOp(char c){ return (int)OP.find(c) >= 0; }
inline int Val(char c) { return c - 'a'; }
void SynErr(int e, char c) {
```

```

cout << "\n Syntax error " << e << ": ";
switch(e) {
    case 1: cout << "unmatched symbol " << c; break;
    case 2: cout << "unexpected symbol " << c; break;
} // switch
exit(0);
}
// Bac uu tien
int Deg(char c) {
    switch(c) {
        case '>':
        case '<':
        case '~':
        case '!': return 100; break;
        case '*':
        case '/':
        case '%': return 200; break;
        case '+':
        case '-': return 300; break;
        default: return 1000;
    } // switch
}
string Compile(string e) {
    cout << "\n Compile " << e;
    string s = "";
    // e->s delete all BL
    s += SHAP; // guard
    for (auto c:e) if (c != BL) s += c;
    string pol = "";
    // stack of operators
    char st[s.length()];
    int p = -1; // index of head of st
    st[++p] = SHAP; // guard
    for (int i = 1; i < s.length(); ++i) {
        char c = s[i]; // xet ky tu s[i]
        // neu c la bien thi dua ra pol
        if (IsVar(c)) { pol += c; continue; }
    }
}

```

46 CHUYÊN TIN 11: CÁC CHUYÊN ĐỀ CHỌN LỌC VỚI NGÔN NGỮ C++

```
// neu c la '(' thi nap st
if (c == '(') { st[++p] = c; continue; }
// neu c la ')':
if (c == ')') {
    // do cac Op tren st ra pol
    while (IsOp(st[p])) pol += st[p--];
    // het st ma chua gap '('
    if (st[p] == SHAP) SynErr(1, c);
    if (st[p] == '(') --p;
    continue;
}
if (IsOp(c)) {
    if (c == '+' || c == '-') {
        if (s[i-1] == SHAP || s[i-1] == '(') {
            if (c == '-') c = '~';
            else continue;
        }
    }
    // Ro cac Op tren st Deg <= Dec(c) ra pol
    while (Deg(st[p]) <= Deg(c)) pol += st[p--];
    st[++p] = c; // Nap c vào st
    continue;
}
SynErr(2, c);
}
// Ro not cac op tren st ra pol
while (IsOp(st[p])) pol += st[p--];
if (p > 0) SynErr(2, st[p]);
return pol;
}
// Factorial n!
long Fac(int n) {
    long r = 1;
    for (int i = 2; i <= n; ++i) r *= i;
    return r;
}
void RunErr(int e, int c) {
    cout << "\n Run error " << e << ": ";
```

```

switch(e) {
    case 1: cout << "divide by zero " << c; break;
    case 2: cout << "negative parameter " << c; break;
} // switch
exit(0);
}

void Exe(string pol) {
    cout << "\n Input Postfix expression: " << pol;
    long st[pol.length()]; // stack
    int p = -1;
    for(auto c:pol) {
        if (IsVar(c)) {st[++p] = Val(c); continue;}
        switch(c) {
            case '~': st[p] = -st[p]; break;
            case '>': ++st[p]; break;
            case '<': --st[p]; break;
            case '!': if (st[p] < 0) RunErr(2, st[p]);
                    st[p] = Fac(st[p]);break;
            case '+': st[p-1] += st[p]; --p; break;
            case '-': st[p-1] -= st[p]; --p; break;
            case '*': st[p-1] *= st[p]; --p; break;
            case '/': if (st[p] == 0) RunErr(1, st[p-1]);
                    st[p-1] /= st[p]; --p; break;
            case '%': if (st[p] == 0) RunErr(1, st[p-1]);
                    st[p-1] %= st[p]; --p; break;
        } // switch
    } // for
    cout << " : " << st[p];
}

int main() {
    string s = "c+d*k"; // 2+3*10 = 32
    string pol = Compile(s);
    cout << " -> " << pol;
    Exe(pol);
    s = "d*d-c*d*h+h*h"; // -> dd*cd*h*-hh*+ : 16
    pol = Compile(s);
    cout << " -> " << pol;
    Exe(pol);
}

```

48 CHUYÊN TIN 11: CÁC CHUYÊN ĐỀ CHỌN LỌC VỚI NGÔN NGỮ C++

```
s = "c-d>!+k*d"; // -> cd>!-kd*+ : 8
pol = Compile(s);
cout << " -> " << pol;
Exe(pol);
s = "c + d*k - (c + d*k) "; // 32 -32 = 0
pol = Compile(s);
cout << " -> " << pol;
Exe(pol);
s = "(k-c*d)-(c+(d+b*e))"; // -> kcd*-cdbe*++- : -5
pol = Compile(s);
cout << " -> " << pol;
Exe(pol);
s = "-(e-b)"; // -(4-1) = -3
// '-(e-b)
pol = Compile(s);
cout << " -> " << pol;
Exe(pol);
s = "-(-e-b)"; // -(-4-1) = 5
pol = Compile(s);
cout << " -> " << pol;
Exe(pol);
cout << "\n T h e   E n d";
return 0;
}
```

Output

```
Compile c+d*k -> cdk*+
Input Postfix expression: cdk*+ : 32
Compile d*d-c*d*h+h*h -> dd*cd*h*-hh*+
Input Postfix expression: dd*cd*h*-hh*+ : 16
. . .
T h e   E n d
```

Sinh dữ liệu test

☞ Tạo một biểu thức đơn giản tùy ý, ví dụ $s = "c+d*k"$;

☞ Tính trị của s (32)

☞ Phức tạp hóa

```
s+s = "(c+d*k) + (c+d*k) "; // 64
```

```
s-s = "(c+d*k) - (c+d*k) "; // 0
```

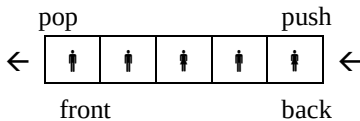
```

s*s = "(c+d*k) * (c+d*k) "; // 1024
s*s = "(c+d*k) / (c+d*k) "; // 1
s%s = "(c+d*k) % (c+d*k) "; // 0
s+s*s = "(c+d*k) + (c+d*k) * (c+d*k) "; // 1056
. . .

```

Queue

Hàng đợi (queue) là cấu trúc dữ liệu được xử lý theo nguyên tắc *vào trước ra trước* (FIFO: First In, First Out) đúng như khi ta xếp hàng chờ được phục vụ. Mỗi hàng đợi q khi đó sẽ có hai cửa phân biệt: cửa vào và cửa ra. Người ta thường quy ước cửa vào ở đầu phải, cửa ra ở đầu trái.



Các thao tác cơ bản trên queue

Khai báo kiểu hàng đợi với các phần tử thuộc kiểu T : `typedef queue<T> QT;`

Khai báo biến hàng đợi q : `QT q;`

Nạp trị v (vào đầu phải) cho q : `q.push(v);`

Bỏ trị (đầu trái) khỏi q : `q.pop();`

`q.size()`: return số phần tử trong q

`q.empty()`: return 1 (true) nếu q rỗng; 0 (false_) nếu q không rỗng

`q.front()`: trị của phần tử trái nhất của q , giữ nguyên q , sinh lỗi nếu q rỗng.

`q.back()`: trị của phần tử phải nhất của q , giữ nguyên q , sinh lỗi nếu q rỗng.

Hiển thị

```

// Print with message msg
void Print(QT q, const char* msg = "") {
    cout << msg;
    while(!q.empty()) {
        cout << " " << q.front();
        q.pop();
    }
}

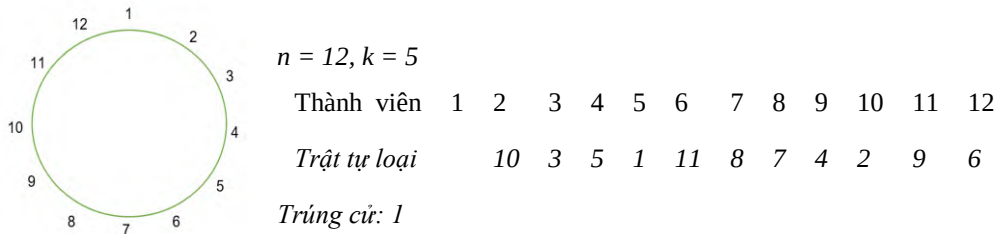
```


Josephus Problem

Trò chơi chọn duy nhất một tù trưởng trong số n người có tài ngang nhau được thực hiện như sau:

Xếp vòng tròn n người. Bắt đầu từ người thứ nhất, đếm k người, loại khỏi vòng tròn người thứ k . Khi nào còn lại 1 người thì người đó được bầu làm tù trưởng.

Ví dụ

**Algorithm****Cách giải thứ nhất**

Mô phỏng qua dãy số. Gọi a là dãy số $1..n$. Duyệt a theo vòng tròn gặp người thứ k thì loại bằng phép gán trị $a[v] = 0$. Độ phức tạp $O(kn)$

Cách giải thứ 2

Mô phỏng qua hàng đợi q . Gọi q là hàng đợi với các giá trị khởi đầu $1..n$. Duyệt q để chuyển $k-1$ người từ đầu q về cuối, loại phần tử thứ k khỏi q . Người cuối cùng còn lại trong q là kết quả. Mô hình hóa theo hàng đợi có tính trực quan hơn cách giải thứ nhất về thao tác loại bỏ phần tử. Giả sử ta có vòng tròn $n = 12$ người và ta phải loại bỏ người thứ $k = 5$ sau mỗi lượt đếm theo bước 5. Ta có:

Khởi trị: $q = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]$

Lượt đếm thứ nhất $q = [6, 7, 8, 9, 10, 11, 12, 1, 2, 3, 4]$ (Loại người số 5)

Lượt đếm thứ hai $q = [11, 12, 1, 2, 3, 4, 6, 7, 8, 9]$ (Loại người số 10)

...

Độ phức tạp: $O(kn)$

Program

```
// Josephus Problem JP, using queue
#include<bits/stdc++.h>
using namespace std;
typedef queue<int> QI;
// Print q with message msg
void Print(QI q, const char* msg = "") {
    cout << msg;
```

```

        while(!q.empty()) { cout << " " << q.front(); q.pop(); }
    }
    int JP(int n, int k) {
        if (n == 1 || k == 1) return n;
        QI q;
        for (int i = 1; i <= n; ++i) q.push(i);
        Print(q, "\n Init: ");
        for (int i = 1; i < n; ++i) {
            // move k-1 elements to the end of q
            for (int j = 1; j < k; ++j) {
                int x = q.front();
                q.pop();
                q.push(x); // ins x to the end
            } // for j
            q.pop(); // loại nguoi thu k
            Print(q, "\n q: ");
        } // for i
        return q.front();
    }
    int main() {
        JP(12, 5);
        cout << "\n T h e   E n d";
        return 0;
    }

```

Output

```

Init: 1 2 3 4 5 6 7 8 9 10 11 12
q: 6 7 8 9 10 11 12 1 2 3 4
q: 11 12 1 2 3 4 6 7 8 9
q: 4 6 7 8 9 11 12 1 2
q: 11 12 1 2 4 6 7 8
q: 6 7 8 11 12 1 2
q: 1 2 6 7 8 11
q: 11 1 2 6 7
q: 11 1 2 6
q: 1 2 6
q: 6 1
q: 1
T h e   E n d

```

Bình luận 1

Một phương án giải khác dựa trên hàm đệ quy như sau. Gọi $V(n, k)$ là số hiệu của người còn lại khi thực hiện trò chơi Josephus với n người và bước đếm k . Ta thấy:

nếu $n = 1$ thì $V(n, k) = 1$

nếu $n > 1$ thì $V(n, k) = (V(n-1, k) + (k-1)) \% n + 1$

Bạn có thể cài đặt phương án này qua hàm lặp Rest như sau:

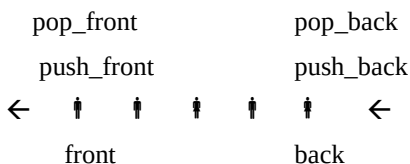
```
int Rest(int n, int k) { // return the rest element
    if (n == 1) return 1;
    int v = 1;
    --k;
    for (int i = 2; i <= n; ++i) v = ((v + k) % i) + 1;
    return v;
}
```

Deque

Hàng đợi hai đầu (deque) là cấu trúc dữ liệu được xử lý theo nguyên tắc hàng đợi (FIFO) theo cả hai đầu, nghĩa là ta có thể:

✂ nạp dữ liệu vào đầu trái, lấy dữ liệu ra khỏi đầu phải của q , hoặc

✂ nạp dữ liệu vào đầu phải, lấy dữ liệu ra khỏi đầu trái của q tùy theo yêu cầu của bài toán.

*Các thao tác cơ bản trên deque*

```
typedef deque<T> DQT; // khai báo kiểu
typedef DQT::iterator IDQT; // khai báo kiểu iterator
DQT dq; // khai báo biến
dq.empty(); // rỗng ?
dq.size(); // size
dq.push_back(x); dq.push_front(x); // nạp x vào đầu phải, đầu trái
dq.pop_front(); dq.pop_back(); // bỏ đầu trái, đầu phải
x = dq.front(); x = dq.back(); // xem trị đầu trái, đầu phải
```

Josephus with Multikey

Chúng ta mở rộng bài Josephus như sau. Thay vì chọn một giá trị k để xác định người thứ k trong vòng tròn sẽ bị loại khỏi vòng tròn ta xét một dãy số nguyên key gọi là khóa. Mỗi lần thực hiện phép đếm-loại ta lấy tuần tự theo vòng tròn một giá trị k trong

dãy khóa *key*. Nếu giá trị k là số dương thì ta sẽ đếm theo chiều kim đồng hồ $k-1$ vị trí, loại người thứ k khỏi vòng tròn rồi tạm dừng tại vị trí kế tiếp theo chiều kim đồng hồ. Ngược lại, khi $k < 0$ thì ta sẽ đếm ngược chiều kim đồng hồ $k-1$ vị trí, loại người thứ k khỏi vòng tròn rồi tạm dừng tại vị trí kế tiếp ngược chiều kim đồng hồ. Người cuối cùng còn lại sẽ là người thắng cuộc.

Algorithm

Với một deque dq ta giải bài này như sau:

Init: $dq = [1, 2, \dots, n]$

lặp $n-1$ lần:

lấy khóa k tiếp theo trong *key*

Nếu $k > 0$:

chuyển $k-1$ người từ đầu về cuối

xóa phần tử thứ k (tại đầu)

Nếu $k < 0$:

$k = \text{abs}(k)-1$

chuyển $k-1$ phần tử cuối dq về đầu

xóa phần tử cuối dq

chuyển 1 pt cuối dq về đầu làm điểm tạm dừng

Minh họa $n = 8, \text{key} = [3, -4, 2, -3]$

Init $dq = [1, 2, 3, 4, 5, 6, 7, 8]$

$k=3$: $dq = [4, 5, 6, 7, 8, 1, 2]$, xóa 3

$k=-4$: $dq = [7, 1, 2, 4, 5, 6]$, xóa 8

$k=2$: $dq = [2, 4, 5, 6, 7]$, xóa 1

$k=-3$: $dq = [5, 7, 2, 4]$, xóa 6

$k=3$: $dq = [4, 5, 7]$, xóa 2

$k=-4$: $dq = [7, 5]$, xóa 4

$k=2$: $dq = [7]$, xóa

Program

```
// Josephus with multi key
#include<bits/stdc++.h>
using namespace std;
typedef deque<int> DQI;
typedef vector<int> VI;
// Print q with message msg
void Print(DQI dq, const char* msg = "") {
    cout << msg;
    while(!dq.empty()) {
        cout << " " << dq.front();
```

54 CHUYÊN TIN 11: CÁC CHUYÊN ĐỀ CHỌN LỌC VỚI NGÔN NGỮ C++

```

        dq.pop_front();
    }
}

int MJP(int n, VI key) {
    if (n == 1) return n;
    int m = key.size();
    cout << "\n Init n = " << n;
    DQI dq;
    for (int i = 1; i <= n; ++i) dq.push_back(i);
    Print(dq, "\n Init q: ");
    int j = -1, k; // j: index for key
    while(dq.size() > 1) { // loop n-1 times
        j = (j+1) % m; k = key[j];
        cout << "\n k = " << k;
        if (k == 0) continue;
        if (k > 0) {
            // chuyen k-1 elems dau -> cuoi
            for(int i = 1; i < k; ++i) {
                dq.push_back(dq.front());
                dq.pop_front(); // loai bo dau
            }
            dq.pop_front(); // loai nguoi thu k
        }
        else { // k < 0
            k = abs(k) - 1;
            for(int i = 1; i < k; ++i) {
                // chuyen k-1 elems cuoi -> dau
                dq.push_front(dq.back());
                dq.pop_back(); // loai bo cuoi
            }
            dq.pop_back(); // loai bo elem thu cuoi
            // chuyen elem cuoi -> dau
            dq.push_front(dq.back());
            dq.pop_back(); // loai bo cuoi
        }
        Print(dq, ", q: ");
    } // while
    return dq.back();
}

```

```

int main() {
    VI key = {3, -4, 2, -3};
    int v = MJP(8, key);
    cout << "\n Result: " << v;
    cout << "\n T h e   E n d";
    return 0;
}

```

Output

```

Init n = 8
Init q:  1 2 3 4 5 6 7 8
k = 3, q:  4 5 6 7 8 1 2
k = -4, q:  7 1 2 4 5 6
k = 2, q:  2 4 5 6 7
k = -3, q:  5 7 2 4
k = 3, q:  4 5 7
k = -4, q:  7 5
k = 2, q:  7
Result: 7
T h e   E n d

```

Độ phức tạp: $O(kn)$

Exercise

Cho biết người bị loại thứ i trong trò chơi Josephus n , k là người nào?

Josephus Cypher

Điều thú vị là bài Josephus with multikey gợi ý cho chúng ta xây dựng một loại mật mã mới. Mỗi loại mật mã gồm hai hàm: mã hóa và giải mã.

Hàm mã hóa $\text{Encode}(s, \text{key})$ nhận vào một string cần mã hóa theo dãy khóa key . Hàm cho ra string w là dãy các ký tự bị xóa khỏi s trong trò chơi Josephus with multikey. Ta gọi s là *bản rõ*, w là *bản mật*.

Hàm giải mã $\text{Decode}(w, \text{key})$ nhận vào bản mật w và dãy khóa key . Hàm cho ra bản rõ s . Để thấy hai hàm này thực hiện các thao tác xuôi-ngược nhau.

Vậy ta chỉ cần một hàm $\text{JP}(s, \text{key}, \text{code}=1)$ với tham biến ngầm định code bao quát cả hai trường hợp. Nếu $\text{code} = 1$ thì ta có hàm Encode ; ngược lại, khi $\text{code} = 0$ thì ta có hàm Decode .

Hàm $\text{JP}(s, \text{key})$ sẽ xếp các đối tượng trong s thành vòng tròn và lần lượt đếm theo khóa k trong key để chuyển mỗi đối tượng cần loại khỏi vòng tròn sang string w .

56 CHUYÊN TIN 11: CÁC CHUYÊN ĐỀ CHỌN LỌC VỚI NGÔN NGỮ C++

- ✧ Nếu $\text{code} = 1$ ta thực hiện mã hóa như thuật toán JP với input là bản rõ s , output là bản mã w , khi cần loại bỏ ký tự $s[i]$ ta chuyển $s[i]$ sang $w[k]$.
- ✧ Nếu $\text{code} \neq 1$, ta thực hiện mã hóa như thuật toán JP với input là bản mã s , output là bản rõ w , khi cần "loại bỏ" ký tự $s[i]$ ta chuyển ngược $w[k]$ sang $s[i]$.

Program

```
// Josephus Cypher with multi key
#include<bits/stdc++.h>
using namespace std;
typedef deque<int> DQI;
typedef vector<int> VI;
// Print q with message msg
void Print(DQI dq, const char* msg = "") {
    cout << msg;
    while(!dq.empty()) {
        cout << " " << dq.front();
        dq.pop_front();
    }
}

string JP(string s, VI key, int code = 1) {
    int n = s.length(), m = key.size();
    if (n == 1 || m == 0) return s;
    string w = s;
    int id, ii = -1, j = -1, k; // j: index for key
    DQI dq;
    for (int i = 0; i < n; ++i) dq.push_back(i);
    while(dq.size() > 1) { // loop n-1 times
        ++ii;
        j = (j+1) % m; k = key[j];
        if (k == 0) continue;
        if (k > 0) {
            // chuyen k-1 elems dau -> cuoi
            for(int i = 1; i < k; ++i) {
                dq.push_back(dq.front());
                dq.pop_front(); // loai bo dau
            }
            id = dq.front(); // element of s must be send to w
            dq.pop_front(); // chuyen
        }
    }
}
```

```

else { // k < 0
    k = abs(k) - 1;
    for(int i = 1; i < k; ++i) {
        // chuyen k-1 elems cuoi -> dau
        dq.push_front(dq.back());
        dq.pop_back(); // loai bo cuoi
    }
    id = dq.back(); // element of s must be send to w
    dq.pop_back(); // xoa cuoi
    // chuyen elem cuoi -> dau
    dq.push_front(dq.back());
    dq.pop_back(); // loai bo cuoi
}
if(code == 1) w[ii] = s[id]; // Encode
else w[id] = s[ii]; // Decode
} // while
id = dq.back(); // element of s must be send to w
if (code) w[n-1] = s[id]; // Encoce
else w[id] = s[n-1]; // Decode
return w;
}
int main() {
    VI key = {3, -4, 2, -3};
    string s, w, u;
    s = "We are learning IT of klass K11 in our school.";
    s += "\n The lesson is very interesting!";
    cout << "\n Input: " << s;
    w = JP(s, key, 1);
    cout << "\n Encoding: " << w;
    u = JP(w, key, 0);
    cout << "\n Decoding: " << u;
    cout << "\n T h e   E n d";
    return 0;
}

```

Output

```

Input: We are learning IT of klass K11 in our school.
The lesson is very interesting!
Encoding: !Wnetiegerns irtvesy io sslne Te

```


58 CHUYÊN TIN 11: CÁC CHUYÊN ĐỀ CHỌN LỌC VỚI NGÔN NGỮ C++

lhooc. suh oir1 Kns a1klfs oI g iTnnen lraea

Decoding: We are learning IT of klass K11 in our school.

The lesson is very interesting!

T h e E n d

Độ phức tạp: $O(kn)$

Map

Map (ánh xạ) là kiểu dữ liệu thiết lập tương ứng giữa các đối tượng thuộc hai thành phần: key (khóa) và value (giá trị). Trong map, giá trị khóa là duy nhất.

Ví dụ, mỗi học sinh (HS) được gán một mã số sẽ được mô tả như một map <int, string>, trong đó:

✂ mã số thuộc kiểu int

✂ tên học sinh thuộc kiểu string

Danh sách học sinh lớp 10A khi đó sẽ bao gồm:

1: Lê Việt Anh

2: Vũ Văn Bình

...

Việc mô tả cụ thể là như sau:

```
typedef map<int, string> MIS; // định kiểu
MIS lop10A, lop10B; // hai biến chứa danh sách HS 10A và 10B
// Nạp dữ liệu
lop10A[1] = "Lê Việt Anh"
lop10A[2] = "Vũ Văn Bình"
...
```

Như vậy, mỗi phần tử trong map sẽ gồm hai thành phần:

first thể hiện mã số HS

last thể hiện tên HS

Khác với kiểu mảng và vector, việc truy nhập đến các phần tử trong kiểu map được quản lý qua *con trỏ* hay *tham chiếu* (iterator thường được hiểu là *kiểu lặp*).

Iterator

Mỗi biến map m có hai giá trị iterator đặc biệt là m.begin() và m.end() trỏ tới phần tử đầu và cuối của m.

Như vậy, nếu m có 10 phần tử thì m.begin() trỏ tới phần tử thứ nhất còn m.end() sẽ trỏ tới phần tử sát sau phần tử thứ 10, tức là 11. Muốn truy nhập phần tử cuối của map m, ta phải viết --m.end().

Nếu t trỏ tới phần tử của map m thì t->first sẽ là khóa, t->second sẽ là trị.

m	(1:An)	(2:Bac)	...	(10.Vinh)	
	m.begin()			m.begin()+10	m.end()
				m.end()-1	

Minh họa

```
#include<bits/stdc++.h>
using namespace std;
typedef map<int, string> MIS; // khai bao kieu map
typedef MIS::iterator IMIS; // khai bao kieu tro map
// Hien thi danh sach lop
void Print(MIS m, const char* msg = "") {
    cout << msg;
    for (auto x:m)
        cout << x.first << " : " << x.second << endl;
}
int main () {
    MIS lop10A;
    lop10A[1] = "Le Viet Anh";
    lop10A[2] = "Vu Van Binh";
    lop10A[3] = "Nguyen Dieu Han";
    Print(lop10A, "\n HS: \n");
    cout << "\n ----- \n";
    IMIS it = lop10A.end();
    --it;
    cout << "\n HS cuoi: " << it->first << " : " << it->second << endl;
    it = lop10A.begin();
    cout << "\n HS dau: " << it->first << " : " << it->second << endl;
    cout << "\n T h e   E n d";
    return 0;
}
```

Output

```
HS:
1 : Le Viet Anh
2 : Vu Van Binh
3 : Nguyen Dieu Han
-----
```

60 CHUYÊN TIN 11: CÁC CHUYÊN ĐỀ CHỌN LỌC VỚI NGÔN NGỮ C++

HS cuối: 3 : Nguyen Dieu Han

HS đầu: 1 : Le Viet Anh

T h e E n d

Chú ý

1. Key trong map không nhất thiết phải là kiểu số, có thể là kiểu sắp được tùy ý: string, char,...
2. Key và value trong map có thể cùng kiểu.

Các thao tác trên map

Khai báo kiểu map. $O(1)$: `typedef map<P, Q> MPQ;`

Ví dụ `typedef map<string, int> MSI;`

Khai báo biến kiểu map. $O(1)$: `MPQ m;`

Khai báo kiểu con trỏ cho MPQ dùng để duyệt các phần tử. $O(1)$

`typedef MPQ::iterator IMPQ;`

`IMPQ it; hoặc MPQ::iterator it;`

Nạp trị v cho khóa k . $O(\log n)$, n là số phần tử trong map

`m.insert(k, v); hoặc m[k] = v;`

ví dụ `m["FOR"] = 1; hoặc m.insert("FOR", 1);`

Kiểm tra khóa k có trong map ? `m.count(k); // 1 if k in m else 0`. $O(\log n)$

Xóa (loại bỏ) phần tử (theo khóa) trong map. $O(\log n)$

`m.erase(k);` xóa phần tử có khóa k

`m.erase(it);` xóa phần tử được trỏ bởi iterator it

`m.erase(it1, it2);` xóa các phần tử được trỏ từ $it1$ đến $it2$

`m.find(k);` return con trỏ tới phần tử có khóa k

`m.emplace(k, v);` tương đương `m[k] = v`.

`m.size();` số phần tử

`m.clear();` hóa rỗng nội dung, giữ lại biến

Words

File WORDS.INP chứa các từ tiếng Anh viết HOA cách nhau và kết thúc bằng dấu chấm (.). Hãy cho biết có bao nhiêu từ khác nhau và mỗi từ xuất hiện bao nhiêu lần.

Output file WORDS.OUT :

Dòng đầu tiên: số lượng từ

Từ dòng thứ hai trở đi: mỗi dòng hai giá trị w m

w là từ có trong input file, m là số lần xuất hiện của từ w trong input file.

WORDS.INP	WORDS.OUT
AND BEGIN END AND END	7
WHILE REPEAT BEGIN DO	AND 8
FOR AND BEGIN END AND END	BEGIN 7
WHILE REPEAT BEGIN DO	DO 3
AND BEGIN END AND END	END 8
FOR AND BEGIN END AND END	FOR 2
WHILE REPEAT BEGIN DO	REPEAT 3
.	WHILE 3

Algorithm

Bài này có thể giải theo hai cách:

Cách thứ nhất: Dùng vector hoặc list hoặc array

- ✂ Đọc lần lượt các từ w trong input file, nạp vào `vector<string> v`. $O(n)$, n là số lượng từ.
- ✂ Sắp tăng v . $O(kn\log(n))$, k là chiều dài max của từ
- ✂ Đếm số lượng các từ khác nhau trong v . $O(n)$
- ✂ Ghi kết quả vào output file. $O(n)$

Program

```
// words.cpp. Version 1: using vector
#include<bits/stdc++.h>
using namespace std;
typedef vector<string> VS;
const char * FN = "WORDS.INP";
const char * GN = "WORDS.OUT";
// delete all c in s
string Trim(string s, char c = 32) {
    string w = "";
    for (auto e:s) if (e != c) w += e;
    return w;
}
// vs: vector sorted string
void Output(VS v){
    ofstream g(GN);
    if(!g) {cout << "\n unable open output file " << GN; exit(0);}
    if (v.size() == 0) { g.close(); return; }
```

62 CHUYÊN TIN 11: CÁC CHUYÊN ĐỀ CHỌN LỌC VỚI NGÔN NGỮ C++

```
v.push_back("."); // guard
string s = v[0];
// count all different words c
int c = 0;
for(auto x:v) if (x != s) { ++c; s = x; }
g << c << endl;
// count word w:c
s = v[0]; c = 0;
for(auto x:v) {
    if (x != s) {g << s << " " << c << endl;s = x; c = 1;}
    else ++c;
}
g.close();
}
void Words() {
    VS v;
    ifstream f(FN);
    if(!f) { cout << "\n unable open input file " << FN; exit(0); }
    string s;
    while (true) {
        f >> s; s = Trim(s);
        if (s == ".") break;
        v.push_back(s);
    }
    f.close();
    sort(v.begin(), v.end());
    Output(v);
}
int main() {
    Words();
    cout << "\n T h e   E n d";
    return 0;
}
```

Cách thứ hai: Dùng map

☞ Đọc lần lượt các từ w trong input file, nạp vào map dạng $\langle \text{key:val} \rangle$ trong đó key là từ w , val là số lần xuất hiện của từ w . $O(n \log n)$

☞ Ghi kết quả vào output file

Độ phức tạp: $O(n \log n)$

Program

```

// words.cpp. Version 2: using map
#include<bits/stdc++.h>
using namespace std;
typedef map<string, int> MSI;
const char * FN = "WORDS.INP";
const char * GN = "WORDS.OUT";
// delete all c in s
string Trim(string s, char c = 32) {
    string w = "";
    for (auto e:s) if (e != c) w += e;
    return w;
}
void Output(MSI d){
    ofstream g(GN);
    if(!g) { cout << "\n unable open output file " << GN; exit(0); }
    g << d.size() << endl;
    for (auto e:d) g << e.first << " " << e.second << endl;
    g.close();
}
void Words() {
    MSI d;
    ifstream f(FN);
    if(!f) { cout << "\n unable open input file " << FN; exit(0); }
    string s;
    while (true) {
        f >> s; s = Trim(s); if (s == ".") break;
        if (d.count(s)) ++d[s]; // khoa s da co
        else d[s] = 1; // s moi duoc them
    }
    f.close(); Output(d);
}
int main() {
    Words();
    cout << "\n T h e   E n d";
    return 0;
}

```

Chú ý

key trong map C++ được sắp tăng, tuy nhiên trong các ngôn ngữ khác không đòi hỏi yêu cầu này.

Farey

John Farey Sr.
(1966-1826)

Cho số nguyên dương n , dãy Farey $F(n)$ được tạo bởi các phân số tối giản $\frac{t}{m}$, $0 \leq t \leq m$, $1 \leq m \leq n$ và được sắp tăng

Ví dụ, với $n = 5$ ta có dãy Farey $F(5)$ gồm 11 phân số tối giản sau đây:

$$F(5) = \frac{0}{1}, \frac{1}{5}, \frac{1}{4}, \frac{1}{3}, \frac{2}{5}, \frac{1}{2}, \frac{3}{5}, \frac{2}{3}, \frac{3}{4}, \frac{4}{5}, \frac{1}{1}$$

Với mỗi số nguyên dương n , hãy tạo sinh dãy $F(n)$?

Algorithm

Chúng ta khảo sát một vài phương án giải bài Farey nhằm mục đích hiểu rõ thêm về các kiểu dữ liệu và phương thức xử lý tương ứng.

Version 1. Using vector

✂ Tạo một lớp phân số (PS) Frac gồm hai trường dữ liệu là Num (Numerator: tử số), Den (Denominator: mẫu số) và các phương thức Frac: khởi trị, Reduce: rút gọn (giản ước) PS, Print: hiển thị PS

✂ Sinh các PS tối giản có mẫu số $m = 1..n$, tử số $t = 1..m$ nạp vào vector d . Độ phức tạp $O(n^2)$

✂ Sắp tăng d . Độ phức tạp $O(n^2 \log n)$

✂ Lược bỏ các PS trùng lặp. Độ phức tạp $O(n^2)$

✂ Độ phức tạp thuật toán $O(n^2 \log n)$

Để rút gọn PS a/b ta tìm $d = \text{uớc chung lớn nhất của } \text{abs}(a) \text{ và } b$ rồi chia a và b cho d .

Muốn so sánh hai PS a/b và c/d ta so sánh tích chéo ad và bc

$$a/b \leq c/d \text{ khi và chỉ khi } ad \leq bc$$

Program

```
// Farey.CPP. Version 1: using vector
#include<bits/stdc++.h>
using namespace std;
int Gcd(int a, int b) { return (b == 0) ? a : Gcd(b, a % b); }
// chuyen doi so nguyen sang string
string Str(int x) {
    stringstream ss; ss << x;
```

```

    string s; ss >> s;
    return s;
}
class Frac {
public:
    int Num; // Tu so
    int Den; // mau so
    // constructors
    Frac() { Num = 0; Den = 1; }
    Frac(int n) { Num = n; Den = 1; }
    Frac(int n, int d) {
        if (d <= 0) { cerr << "\n ERROR: Mau so <= 0: " << d ; exit(0); }
        Num = n; Den = d;
    }
    string ToStr() {
        string s = Str(Num);
        return (Den == 1 || Num == 0) ? s : s += "/" + Str(Den);
    }
    void Print(const char * msg = "") { cout << msg << ToStr(); }
    Frac Reduce() { int d = Gcd(abs(Num), Den); return Frac(Num / d, Den / d); }
};

typedef vector<Frac> VF;
// Compares two intervals according to starting times.
bool Less(Frac x, Frac y) { return (x.Num*y.Den < x.Den*y.Num); }
bool Eq(Frac x, Frac y) { return (x.Num*y.Den == x.Den*y.Num); }
VF Farey(int n) {
    VF d;
    d.push_back(Frac(0)); d.push_back(Frac(1)); // 0/1, 1/1
    for(int mau = 2; mau <= n; ++mau)
        for(int tu = 1; tu < mau; ++tu)
            d.push_back(Frac(tu, mau).Reduce());
    for(auto x:d) x.Print(" "); cout << "\n Total " << d.size();
    sort(d.begin(), d.end(), Less);
    cout << "\n sorted d: ";
    for(auto x:d) x.Print(" "); cout << "\n Total " << d.size();
    // eliminate duplicated fractions
    VF r; // the result
    Frac y(1,n+1);

```


66 CHUYÊN TIN 11: CÁC CHUYÊN ĐỀ CHỌN LỌC VỚI NGÔN NGỮ C++

```
for(auto x:d) if (!Eq(x,y)) { r.push_back(x); y = x; }
return r;
}
int main() {
    VF r = Farey(5);
    cout << "\n Result:";
    for(auto x:r) x.Print(" ");
    cout << "\n Total " << r.size();
    cout << "\n T h e   E n d";
    return 0;
}
```

Output

```
0 1 1/2 1/3 2/3 1/4 1/2 3/4 1/5 2/5 3/5 4/5
Total 12
sorted d:  0 1/5 1/4 1/3 2/5 1/2 1/2 3/5 2/3 3/4 4/5 1
Total 12
Result: 0 1/5 1/4 1/3 2/5 1/2 3/5 2/3 3/4 4/5 1
Total 11
T h e   E n d
```

Version 2. Using map

1. Sinh các phân số (PS) tối giản, nạp vào map d kiểu <double, Frac>, trong đó khóa có kiểu double là thương của tử / mẫu, value chính là PS. d sẽ tự động được sắp tăng theo khóa. Độ phức tạp $O(n^2 \log n)$.
2. Thông báo kết quả là thành phần thứ hai (second) của map d.

Độ phức tạp thuật toán: $O(n^2 \log n)$

Program

```
// Farey.CPP. Version 2: using map
#include<bits/stdc++.h>
using namespace std;
int Gcd(int a, int b) { return (b == 0) ? a : Gcd(b, a % b); }
// chuyen doi so nguyen sang string
string Str(int x) {
    stringstream ss; ss << x;
    string s; ss >> s;
    return s;
}
```

```

class Frac { ... };
typedef map<double, Frac> MDF;
// Hien thi map<Frac> m
void Print(MDF m, const char*msg = "") {
    cout << msg;
    for(auto x:m) (x.second).Print(" ");
    cout << "\n Total: " << m.size();
}
// (double)tu / mau
inline double Val(Frac x) { return (double)x.Num / x.Den; }
void Farey(int n) {
    MDF d;
    Frac x(0), y(1);
    d[Val(x)] = x; d[Val(y)] = y; // 0/1, 1/1
    for(int mau = 2; mau <= n; ++mau) {
        for(int tu = 1; tu < mau; ++tu) {
            x = Frac(tu, mau).Reduce();
            d[Val(x)] = x;
        }
    }
    Print(d, "\n Result: ");
}
int main() {
    Farey(5);
    cout << "\n T h e   E n d";
    return 0;
}

```

Version 3. Using Median Fraction

PS trung bình của hai PS a/b và c/d là PS tối giản của $(a+c)/(b+d)$ tức là tử + tử / mẫu + mẫu, rút gọn.

Tính chất: PS trung bình của hai PS x và y luôn nằm giữa x và y .

Tính chất trên gợi ý cho chúng ta triển khai phương án 3 như sau:

☞ Với mẫu số $m = 1$, khởi tạo hai PS đầu và cuối dãy Farey a là $0/1$ và $1/1$

☞ Lặp với mẫu số $m = 2..n$:

- Tìm hai PS liên tiếp trong a có tổng mẫu số bằng m , tạo PS trung bình của hai PS này đặt vào giữa.

Ví dụ, $n = 5$

$m = 1$: 0/1 1/1

$m = 2$: 0/1 **1/2** 1/1

$m = 3$: 0/1 **1/3** 1/2 **2/3** 1/1

$m = 4$: 0/1 **1/4** 1/3 1/2 **2/3** **3/4** 1/1

$m = 5$: 0/1 **1/5** 1/4 1/3 **2/5** 1/2 **3/5** 2/3 **3/4** **4/5** 1/1

Độ phức tạp: $O(n^2)$

Program

Ta dùng hai vector PS a , b , tại bước lặp thứ m , ta chuyển đổi từ a sang b theo PS trung bình có tổng các mẫu số là m . Tại cuối bước lặp, ta gán $a = b$.

Chú ý rằng với hai vector a và b thì phép gán $a = b$ đòi hỏi độ phức tạp $O(b.size())$.

Ngoài ra, trong class `Frac`, ta cài đặt thêm phép toán lấy PS trung bình của hai PS x và y , ký hiệu là $x \% y$.

```
// Farey.CPP. Version 3: using median fractions
#include<bits/stdc++.h>
using namespace std;
int Gcd(int a, int b) {
    return (b == 0) ? a : Gcd(b, a % b);
}
// chuyen doi so nguyen sang string
string Str(int x) {
    stringstream ss; ss << x;
    string s; ss >> s;
    return s;
}
class Frac {
    . . .
    Frac operator %(const Frac & y) { // median Fraction
        return Frac(Num + y.Num, Den + y.Den).Reduce();
    }
};
typedef vector<Frac> VF;
// Hien thi map PS m
void Print(VF v, string msg = "") {
    cout << msg;
    for(auto x:v)
        x.Print(" ");
    cout << "\n Total: " << v.size();
}
```

```

void Farey(int n) {
    VF a;
    a.push_back(Frac(0)); a.push_back(Frac(1));
    int m = 1;
    Print(a, "\n m = " + Str(m)+ ": ");
    for (m = 2; m <= n; ++m) {
        VF b;
        b.push_back(a[0]);
        for (int i = 1; i < a.size(); ++i) {
            if (a[i-1].Den + a[i].Den == m)
                b.push_back(a[i-1] % a[i]);
            b.push_back(a[i]);
        }
        a = b;
        Print(a, "\n m = " + Str(m) + ": ");
    }
    Print(a, "\n Result: ");
}

int main() {
    Farey(5);
    cout << "\n T h e   E n d";
    return 0;
}

```

Version 4. Directly Generate

Nhận xét: Nếu a, b và c là ba PS liên tiếp trong F(n) thì:

1. $b.Num * a.Den - a.Num * b.Den = 1$
2. $a.Den + b.Den > n$
3. $b = \text{Median}(a, c) // b = a \% c$
4. $c.Num = v * b.Num - a.Num, c.Den = v * b.Den - a.Den$
 $v = (a.Den + n) / b.Den$

Algorithm

Khởi tạo d với hai PS đầu tiên $a = 0/1, b = 1/n$
 Lập đến khi gặp PS cuối cùng $c = 1/1$
 Tính c theo tính chất 4
 $d.push_back(c)$
 $a = b; b = c;$

Độ phức tạp: $O(n^2)$

70 CHUYÊN TIN 11: CÁC CHUYÊN ĐỀ CHỌN LỌC VỚI NGÔN NGỮ C++

Program

```
// Farey.CPP. Version 4: Directly generate
#include<bits/stdc++.h>
using namespace std;
// chuyen doi so nguyen sang string
string Str(int x) {
    stringstream ss; ss << x;
    string s; ss >> s;
    return s;
}
class Frac {
public:
    int Num; // Tu so
    int Den; // mau so
    Frac() { Num = 0; Den = 1; }
    Frac(int n) { Num = n; Den = 1; }
    Frac(int n, int d) {
        if (d <= 0) {
            cerr << "\n ERROR: Mau so <= 0: " << d ;
            exit(1);
        }
        Num = n; Den = d;
    }
    void Set(int n, int d) {
        if (d <= 0) {
            cerr << "\n ERROR: Mau so <= 0: " << d ;
            exit(1);
        }
        Num = n; Den = d;
    }
    string ToStr() {
        string s = Str(Num);
        return (Den == 1 || Num == 0) ? s
            : s += "/" + Str(Den);
    }
    void Print(const char * msg = "") { cout << msg << ToStr(); }
    bool operator ==(const Frac & y) { return Num*y.Den == Den*y.Num; }
    bool operator !=(const Frac & y) { return !(*this == y); }
};
```

```

typedef vector<Frac> VF;
VF Farey(int n) {
    Frac a(0), b(1,n), c;
    const Frac ONE(1);
    VF d;
    d.push_back(a);
    d.push_back(b);
    while (c != ONE) {
        int v = (a.Den + n) / b.Den;
        c.Set(v*b.Num - a.Num, v*b.Den - a.Den);
        d.push_back(c);
        a = b; b = c;
    }
    return d;
}

int main() {
    VF v = Farey(5);
    cout << "\n Result:";
    for(auto x:v) x.Print(" ");
    cout << "\n Total " << v.size();
    cout << "\n T h e   E n d";
    return 0;
}

```

Output

```

Result: 0 1/5 1/4 1/3 2/5 1/2 3/5 2/3 3/4 4/5 1
Total 11
T h e   E n d

```

Version 5. Using List

Algorithm

1. Sinh dãy Farey $F(n)$ ghi vào list $d = F(n)$. $O(n^2)$
2. Sort d . $O(n \log n)$
3. $d.unique()$. $O(n)$
4. Thông báo kết quả.

Độ phức tạp: $O(n^2)$

Program

```

// Farey.CPP. Version 5: Using list
#include<bits/stdc++.h>

```

72 CHUYÊN TIN 11: CÁC CHUYÊN ĐỀ CHỌN LỌC VỚI NGÔN NGỮ C++

```

using namespace std;
int Gcd(int a, int b) {
    return (b == 0) ? a : Gcd(b, a % b);
}
// chuyen doi so nguyen sang string
string Str(int x) {
    stringstream ss; ss << x;
    string s; ss >> s;
    return s;
}
class Frac { . . . };
typedef list<Frac> LF;
// Hien thi vector Frac m
void Print(LF v, string msg = "") {
    cout << msg;
    for(auto x:v)
        x.Print(" ");
    cout << "\n Total: " << v.size();
}
bool Less(Frac a, Frac b) { return a < b; }
void Farey(int n) {
    LF d;
    d.push_back(Frac(0)); d.push_back(Frac(1));
    for(int m = 2; m <= n; ++m) {
        for(int t = 1; t < m; ++t)
            d.push_back(Frac(t,m).Reduce());
    }
    Print(d, "\n Init: ");
    d.sort(Less);
    d.unique();
    Print(d, "\n Result: ");
}
int main() {
    Farey(8);
    cout << "\n T h e   E n d";
    return 0;
}

```

Output

```

Init:  0 1 1/2 1/3 2/3 1/4 1/2 3/4 1/5 2/5 3/5 4/5 1/6 1/3 1/2 2/3 5/6
1/7 2/7 3/7 4/7 5/7 6/7 1/8 1/4 3/8 1/2 5/8 3/4 7/8
Total: 30
Result: 0 1/8 1/7 1/6 1/5 1/4 2/7 1/3 3/8 2/5 3/7 1/2 4/7 3/5
5/8 2/3 5/7 3/4 4/5 5/6 6/7 7/8 1
Total: 23
T h e   E n d

```

Luyện tập Farey

1. Viết hàm $\text{Len}(n)$ cho ra chiều dài của dãy Farey $F(n)$. Ví dụ $\text{Len}(5) = 11$; $\text{Len}(15) = 73$.
2. Viết hàm $\text{Index}(\text{int } n, \text{int } t, \text{int } m)$ cho ra số thứ tự (tính từ 1) của PS t/m trong dãy Farey $F(n)$. Ví dụ $\text{Index}(15, 1, 12) = 5$; $\text{Index}(15, 1, 1) = 73$.
3. Viết hàm $\text{Val}(\text{int } n, \text{int } i)$ cho ra PS thứ i (tính từ 1) trong dãy Farey $F(n)$.
Ví dụ: $\text{Val}(15, 5) = 1/12$; $\text{Val}(15, 73) = 1$.

Gợi ý

- ✂ Viết hàm Index theo Version 4 (directly generate)
- ✂ $\text{Len}(n) = \text{Index}(n, 1, 1)$
- ✂ Val tựa như Index

Bài giải Farey

Đầu tiên xây dựng hàm $\text{Frac Next}(\text{int } n, \text{Frac } a, \text{Frac } b)$ sinh ra PS tiếp theo c từ hai PS liên tiếp a và b trong $F(n)$.

Sau đó triển khai các hàm Index , Len và Val .

Program

```

// Farey Applications
#include<bits/stdc++.h>
using namespace std;
int Gcd(int a, int b) { . . . }
// chuyen doi so nguyen sang string
string Str(int x) { . . . }
class Frac { . . . };
// a, b -> c
Frac Next(int n, Frac a, Frac b) { // O(1)
    int v = (a.Den + n) / b.Den;
    return Frac(v*b.Num - a.Num, v*b.Den - a.Den);
}

```


74 CHUYÊN TIN 11: CÁC CHUYÊN ĐỀ CHỌN LỌC VỚI NGÔN NGỮ C++

```
int Index(int n, int t, int m = 1) { //  $O(n^2)$ 
    Frac x(t,m);
    Frac a(0), b(1,n), c; // c = 0
    if(x == a) return 1;
    if(x == b) return 2;
    int d = 2;
    while (c != x) {
        ++d;
        c = Next(n, a, b); a = b; b = c;
    }
    return d;
}

int Len(int n) { return Index(n, 1); }

Frac Val(int n, int id) {
    Frac a(0), b(1,n), c; // c = 0
    if(id == 1) return a;
    if(id == 2) return b;
    for (int i = 3; i <= id; ++i) {
        c = Next(n, a, b); a = b; b = c;
    }
    return c;
}

int main() {
    int len5 = Len(5);
    cout << "\n Len of F(5) = " << len5;
    cout << "\n Len of F(15) = " << Len(15);
    cout << "\n F(5):";
    for(int id = 1; id <= len5; ++id) {
        Val(5, id).Print(" ");
    }
    cout << "\n T h e   E n d";
    return 0;
}
```

Output

```
Len of F(5) = 11
Len of F(15) = 73
F(5): 0 1/5 1/4 1/3 2/5 1/2 3/5 2/3 3/4 4/5 1
T h e   E n d
```

Class Frac

Theo tiếp cận của lập trình hướng đối tượng mỗi kiểu dữ liệu bao gồm hai thành phần:

☞ D: Cấu trúc dữ liệu

☞ P: Các phương thức xử lý dữ liệu trong D

Các kiểu dữ liệu được tập hợp thành một module độc lập tương đối với nhau.

Các dự án tin học (IT project) bao gồm nhiều module.

Giả sử chúng ta cần giải quyết một số vấn đề liên quan đến các phân số, chẳng hạn:

☞ Xây dựng phần mềm phục vụ cho việc giảng dạy và học tập về phân số trong chương trình Toán học phổ thông

☞ Giải các bài toán về dãy Farey

Chúng ta cần tạo một kiểu dữ liệu mới và đặt tên là Frac. Kiểu Frac sẽ bao gồm 2 thành phần:

$$\text{Frac} = (\text{D}, \text{P})$$

☞ D là cấu trúc dữ liệu biểu diễn một phân số (PS):

(num, den), trong đó num là tử số là một số nguyên, den là mẫu số là một số nguyên dương

☞ P là các phép toán trên phân số:

Cộng hai PS

Trừ hai PS

Nhân hai PS

Chia hai PS

Tạo PS trung bình của hai PS

So sánh hai PS

Hiển thị một PS

Chuyển đổi một PS sang string

Chuyển đổi một PS sang số thực (gần đúng)

OverLoad

Trong lập trình hướng đối tượng, overload được hiểu là *định lại*, nghĩa là sử dụng lại một ký hiệu, một hàm, một tên gọi theo nghĩa khác. Ví dụ, ký hiệu + thường được hiểu là dấu phép toán cộng hai số, nay ta muốn dùng dấu này để ký hiệu cho phép toán cộng hai đối tượng không phải số như hai string hoặc hai ma trận hoặc hai phân số,... thì ta phải vận dụng khái niệm định lại (overload). Khi đó chỉ thị:

76 CHUYÊN TIN 11: CÁC CHUYÊN ĐỀ CHỌN LỌC VỚI NGÔN NGỮ C++

`Frac operator + (const Frac & y) hoặc`

`Frac operator + (const Frac & x, const Frac & y)`

sẽ báo cho hệ thống biết là: Tôi muốn định lại `+` là phép cộng (+) hai phân số. Như vậy, khi bạn viết:

```
Frac x(5, 12);
```

```
Frac y(1, 12);
```

```
z = x + y;
```

Thì hệ thống sẽ hiểu là bạn muốn cộng PS $x = 5/12$ với PS $y = 1/12$ để cho ra PS z .

Xét khai báo:

```
Frac operator +(const Frac & y)
```

Phép cộng hai phân số là phép toán hai ngôi, trong khi đó dòng lệnh trên chỉ có một tham biến thứ hai là `y`. Tham biến thứ nhất được hiểu là chủ thể (`this`) được cộng với `y`, do đó phương thức này cần được đặt trong class `Frac` và được hiểu là:

`this + y` cho ra `Frac`

Mặt khác, khai báo:

```
Frac operator +(const Frac & x, const Frac & y)
```

chỉ rõ hai tham biến `x` và `y` cho phép cộng được đặt ngoài class `Frac` và được coi là hàm bạn (`friend`). Khai báo trên được hiểu như một lời gọi hàm thông thường.

Nếu khai báo trên được đặt trong class thì cần thêm tiếp đầu `friend` (hàm bạn).

Implement of Frac Class

Bạn soạn thảo một file `Frac.h` để cài đặt module cho class `Frac` như sau:

```
/******  
Frac 6 Mar 2025  
*****/  
#ifndef __Frac_H  
#define __Frac_H  
#include<bits/stdc++.h>  
using namespace std;  
/*-----  
CAC CHUC NANG PHUONG THUC PHU TRO  
-----*/  
typedef long long Long;  
Long Gcd(Long a, Long b) { return (b == 0) ? a : Gcd(b, a % b); }  
// Bao loi, dung chuong trinh  
void Err(int e) {  
    cout << "\n Error " << e << ": ";  
}
```

```

switch(e) {
    case 1: cout << " Numerator <= 0."; break;
    case 2: cout << " divide by zero."; break;
}
exit(e);
}
// chuyen doi so nguyen sang string
string Str(Long x) {
    stringstream ss; ss << x;
    string s; ss >> s;
    return s;
}
class Frac {
public:
    Long Num; // Tu so
    Long Den; // mau so
    /*-----
        Constructors
    -----*/
    Frac() { Num = 0; Den = 1; }
    Frac(Long n) { Num = n; Den = 1; }
    Frac(Long n, int d) {
        if (d <= 0L) Err(1);
        Num = n; Den = d;
    }
    Frac(const Frac &y) { Num = y.Num; Den = y.Den; }
    /*-----
        Set: gan tri
    -----*/
    void Set() { Num = 0; Den = 1; }
    void Set(Long n) { Num = n; Den = 1; }
    void Set(Long n, int d) {
        if (d <= 0L) Err(1);
        Num = n; Den = d;
    }
    void Set(const Frac &y) { Num = y.Num; Den = y.Den; }
    Frac Reduce() {
        Long d = Gcd(abs(Num),Den);

```

78 CHUYÊN TIN 11: CÁC CHUYÊN ĐỀ CHỌN LỌC VỚI NGÔN NGỮ C++

```
    Num /= d; Den /= d;
    return *this;
}

string ToStr() const {
    string s = Str(Num);
    return (Den == 1 || Num == 0) ? s : s += "/" + Str(Den);
}

Frac Inv() const { // Frac nghịch đảo t/m -> m/t
    if (Num == 0) Err(2);
    long d = (Num < 0) ? -1 : 1;
    return Frac(d*Den, d*Num);
}

// Chuyen Frac sang so thuc
double Val() { return (double)Num/Den; }

friend Frac operator +(const Frac& x, const Frac& y) {
    return Frac(x.Num*y.Den + x.Den*y.Num, x.Den*y.Den).Reduce();
}

Frac Neg() const { // dao dau cua tu so
    return Frac(-Num, Den);
}

friend Frac operator -(const Frac& x, const Frac& y) {
    return x + y.Neg();
}

friend Frac operator *(const Frac& x, const Frac& y) {
    return Frac(x.Num*y.Num, x.Den*y.Den).Reduce();
}

friend Frac operator /(const Frac& x, const Frac& y) {
    return x*y.Inv();
}

friend Frac operator %(const Frac& x, const Frac& y) { // median
    return Frac(x.Num + y.Num, x.Den + y.Den).Reduce();
}

void operator +=(const Frac & y) { *this = *this + y; }
void operator -=(const Frac & y) { *this = *this - y; }
void operator *=(const Frac & y) { *this = *this * y; }
void operator /=(const Frac & y) { *this = *this / y; }
void operator %=(const Frac & y) { *this = *this % y; }

// F(++x) = ++x; F(x)
```

```

Frac operator ++() { *this += 1; return *this; }
// F(x++) = F(x); x++;
Frac operator ++(int v = 1) {
    Frac y = *this; *this += 1; return y;
}
// F(--x) = --x; F(x)
Frac operator --() { *this -= 1; return *this; }
// F(x--) = F(x); --x;
Frac operator --(int v = 1) {
    Frac y = *this; *this -= 1;
    return y;
}
}; // class Frac
ostream & operator <<(ostream & s, const Frac & x) {
    s << x.ToStr();
    return s;
}
/*-----
So sanh 2 phan so x va y
0: x = y, > 0: x > y, < 0: x < y
-----*/
inline Long Compare(const Frac & x, const Frac & y) {
    return x.Num*y.Den - x.Den*y.Num;
}
// Cac phep so sanh hai phan so
inline bool operator ==(const Frac& x, const Frac& y) {
    return (Compare(x,y) == 0);
}
inline bool operator !=(const Frac& x, const Frac& y) {return !(x == y); }
inline bool operator >(const Frac& x, const Frac& y) {
    return (Compare(x,y)>0); }
inline bool operator >=(const Frac& x, const Frac& y) {
    return (Compare(x,y) >= 0); }
inline bool operator <(const Frac& x, const Frac& y) { return !(x >= y);}
inline bool operator <=(const Frac& x, const Frac& y) {return !(x > y);}
#endif

```

Bạn đặt tên cho file chương trình là Frac.h, ghi file vào thư mục của bạn, compile file này, sau đó thử viết vài test để đánh giá module Frac.

Frac Testing 1

Test 1 thực hiện các thao tác sau đây:

- ✂ Khởi tạo các PS x, y
- ✂ Hiển thị x, y
- ✂ Thực hiện một số phép toán. Lưu ý rằng bạn đã định lại ký hiệu $\%$ là phép tính PS trung bình.
- ✂ Thực hiện các phép so sánh

```
// Test1 Frac
#include<bits/stdc++.h>
#include "Frac.H"
using namespace std;
int main() {
    Frac x(5, 12), y(1, 12), z;
    z = x + y;
    cout << "\n x = " << x;
    cout << "\n y = " << y;
    cout << "\n z = x + y = " << z;
    z = x - y;
    cout << "\n z = x - y = " << z;
    z = x * y;
    cout << "\n z = x * y = " << z;
    z = x / y;
    cout << "\n z = x / y = " << z;
    z = x % y;
    cout << "\n Median z = x % y = " << z;
    cout << "\n x = y ? " << (x == y); // 0
    cout << "\n x != y ? " << (x != y); // 1
    cout << "\n x < y ? " << (x < y); // 0
    cout << "\n x <= y ? " << (x <= y); // 0
    cout << "\n x > y ? " << (x > y); // 1
    cout << "\n x >= y ? " << (x >= y); // 1
    cout << "\n x >= x ? " << (x >= x); // 1
    cout << "\n x > x ? " << (x > x); // 0
    cout << "\n x == x ? " << (x == x); // 1
    cout << "\n x != x ? " << (x != x); // 0
    cout << "\n T h e   E n d";
    return 0;
}
```

Frac Testing 2

Test thứ hai cho module Frac dựa trên *tính thuận nghịch* như sau:

☞ Sinh ngẫu nhiên ba PS: x , y và z

☞ Tính $t = (x + y) * z$ // $t = x * z + y * z$

☞ Tính $v = (t - y * z) / z$ // $v = ((x * z + y * z) - y * z) / z = x * z / z = x$

☞ Kiểm tra $v == x$?

Đặt dãy lệnh trên vào vòng lặp nhiều lần ta có thể đánh giá được tình huống cài đặt module Frac.

Program

```
// Test2 Frac
#include<bits/stdc++.h>
#include "Frac.H";
using namespace std;
inline void Go(const char * msg = " ? ") {
    cout << msg; fflush(stdin);
    if (cin.get() == '.') exit(0);
}
void Test2() {
    srand(time(NULL));
    int r = 20;
    Frac t, v;
    while(true) {
        Frac x(rand() % r, rand() % r + 1);
        Frac y(rand() % r, rand() % r + 1);
        Frac z(rand() % r + 1, rand() % r + 1);
        cout << "z = " << z << endl;
        cout << "x = " << x << endl;
        cout << "y = " << y << endl;
        cout << "z = " << z << endl;
        t = (x + y) * z; // t = (x+y)z = xz + yz
        cout << "t = (x+y)*z = " << t << endl;
        v = (t - y*z) / z;
        cout << "v = (t - y*z) / z = " << v << endl;
        if (v != x) cout << "Error." << endl;
        else cout << "Correct." << endl;
        Go();
    }
}
```


82 CHUYÊN TIN 11: CÁC CHUYÊN ĐỀ CHỌN LỌC VỚI NGÔN NGỮ C++

```
int main() {  
    Test2();  
    cout << "\n T h e   E n d";  
    return 0;  
}
```

Output Test2

```
z = 1/5  
x = 15/5  
y = 9/9  
z = 1/5  
t = (x+y)*z = 4/5  
v = (t - y*z) / z = 3  
Correct.  
?  
z = 19/4  
x = 4  
y = 16/5  
z = 19/4  
t = (x+y)*z = 171/5  
v = (t - y*z) / z = 4  
Correct.  
? .
```

Giải thích thiết kế và các nguyên lý Test

Nguyên lý nhất quán

Nguyên lý này quy các thao tác về cùng một mối.

Ví dụ

Thay phép trừ hai PS, $x-y$ bằng phép cộng $x + y.Neg()$, trong đó $y.Neg()$ là hàm đảo dấu của PS y .

Thay phép chia PS x/y bằng phép nhân $x*y.Inv()$, trong đó $y.Inv()$ là hàm lấy nghịch đảo PS y .

Thay $!=$ bằng phủ định của $==$

Thay $x < y$ bằng $!(x \geq y)$, ...

Khi cần sửa ta chỉ phải sửa lại đầu mỗi, tức là một chỗ.

Nguyên lý thuận nghịch

Nguyên lý này cho phép test tự động các quy trình thuận nghịch

☞ Quy trình biến đổi từ A đến B: $A \rightarrow B$;

☞ Quy trình biến đổi ngược từ B đến C: $B \rightarrow C$;

☞ Nếu $C = A$ tức là đúng.

Nguyên lý đầy đủ

Phải thực hiện test mọi tình huống. Ví dụ $x = x ?$, $x != x ?$, $x \leq x ?$...

Các tham biến với chỉ thị const & x cho biết x được truyền theo địa chỉ và giá trị của x không được phép thay đổi trong hàm.

Chỉ thị inline thông báo cho chương trình dịch tìm cách đặt trực tiếp đoạn mã của hàm trong bản dịch, do đó tránh được thủ tục truyền tham trị nên giảm được thời gian thực hiện.

Tại sao Frac có thể làm việc với các đối tượng kiểu int?

Là vì khi khai báo Frac trong các tạo tử (constructor) và các phép gán trị (Set) chúng ta đã quy định rằng nếu chỉ có một tham biến nguyên là t thì Frac sẽ được gán trị là t/1.

Ví dụ

```
Frac x(1,2);
```

```
Frac y = x + 1 ; // y = 1/2 + 1/1 = 1/2 + 2/2 = 3/2
```

Ngoài ra, chúng ta không cài đặt phép toán +, -, *, /, % (tạm ký hiệu chung là @) dưới dạng this @ y.

```
Frac operator @(const Frac &y)
```

Theo dạng trên thì lời gọi this @y chỉ được thực hiện khi this là Frac.

Tuy nhiên, trong class Frac chúng ta cài đặt các phép toán @ dưới dạng hàm bạn hai ngôi tường minh.

```
friend Frac operator @(const Frac& x, const Frac& y)
```

Do đó, khi thực hiện các phép toán @, cả hai tham biến x và y đều được gán trị bình đẳng nhau, dù là int hay Frac đều được chuyển đổi sang Fac.

$++x$, $x++$, $--x$, $x--$ có gì khác nhau ?

Khi đứng một mình (độc lập) thì $++x$ và $x++$ không có gì khác nhau: chúng đều tăng giá trị của PS x thêm 1 đơn vị.

```
Frac x(1,2);
```

```
cout << "\n Init x = " << x; // 1/2
```

```
++x;
```

```
cout << "\n ++x = " << x; // 3/2
```

```
x.Set(1,2); // gan x tri 1/2
```

```
cout << "\n Set x = " << x; // 1/2
```

```
x++;
```

```
cout << "\n x++ = " << x; // 3/2
```

84 CHUYÊN TIN 11: CÁC CHUYÊN ĐỀ CHỌN LỌC VỚI NGÔN NGỮ C++

Khi đứng độc lập, các phép toán $--x$ và $x--$ cũng tương tự như trên: chúng đều giảm giá trị của PS 1 đơn vị.

```
Frac x(1,2);
cout << "\n Init x = " << x; // 1/2
--x;
cout << "\n --x = " << x; // 1/2 - 1 = 1/2 - 2/2 = -1/2
x.Set(1,2);
cout << "\n Set x = " << x; // 1/2
x--;
cout << "\n x-- = " << x; // 1/2 - 1 = 1/2 - 2/2 = -1/2
```

Khi đặt các phép toán $++$ và $--$ trong *biểu thức* thì tình hình sẽ khác.

Ví dụ: Giả sử bạn cài đặt hàm $F(x)$ là hàm cho ra tích của tử và mẫu của PS x

```
int F(Frac x) {
    return x.Num*x.Den;
}
```

Và PS x được khởi tạo $1/2$

```
Frac x(1,2);
```

Thì hai lời gọi $F(x++)$ và $F(++x)$ sẽ tạo ra các hiệu ứng khác nhau, cụ thể là:

Lời gọi $F(x++)$ sẽ được thực hiện theo hai bước:

☞ $F(x)$

☞ $x++$; // tăng x thêm 1

Do đó:

```
Frac x(1,2); // x = 1/2
F(x++);
// tương đương với
// F(x) = 1*2 = 2
// x++ -> x = 3/2
```

Lời gọi $F(++x)$ sẽ được thực hiện như sau:

☞ $++x$; // tăng thêm 1

☞ $F(x)$

Do đó:

```
Frac x(1,2); // x = 1/2
F(++x);
// tương đương với
// ++x -> x = 3/2
// F(x) = 3*2 = 6
```

Bạn quan sát để thấy ++ được cài đặt trong class theo hai dạng phân biệt như trên, cụ thể là:

```
// F(++x) = ++x; F(x)
Frac operator ++() {
    *this += 1; // tang this them 1
    return *this;
}

// F(x++) = F(x); x++;
Frac operator ++(int v = 1) {
    Frac y = *this; // luu this
    *this += 1; // tang this them 1
    return y; // return this truoc khi tang
}
```

Tương tự, lời gọi $F(--x)$ sẽ được thực hiện như sau:

✂ $--x$; // giảm x bớt 1

✂ $F(x)$

Do đó:

```
Frac x(1,2); // x = 1/2
F(--x);
// tương đương với
// --x -> x-1 = 1/2 - 2/2 = -1/2
// F(x) = (-1)*2 = -2
```

Lời gọi $F(x--)$ sẽ được thực hiện như sau:

✂ $F(x)$

✂ $x--$; // giảm x bớt 1

```
Frac x(1,2); // x = 1/2
F(x--);
// tương đương với
// F(x) = 1 * 2 = 2
// x-- -> x = -1/2
```

Frac Test 3

```
// Test3 Frac ++ and --
#include<bits/stdc++.h>
#include "Frac.H";
using namespace std;
int F(Frac x) { return x.Num*x.Den; }
```

86 CHUYÊN TIN 11: CÁC CHUYÊN ĐỀ CHỌN LỌC VỚI NGÔN NGỮ C++

```
void Test3() {
    Frac x(1,2);
    cout << "\n Init x = " << x; // 1/2
    // ++
    cout << "\n F(x++) = " << F(x++); // 2
    cout << "\n now x = " << x; // 3/2
    cout << "\n F(++x) = " << F(++x); // 10
    cout << "\n now x = " << x; // 5/2
    x.Set(1,2);
    cout << "\n reset x = " << x; // 1/2
    // --
    cout << "\n F(x--) = " << F(x--); // 2
    cout << "\n now x = " << x; // -1/2
    cout << "\n F(--x) = " << F(--x); // -6
    cout << "\n now x = " << x; // -3/2
}
int main() {
    Test3();
    cout << "\n T h e   E n d";
    return 0;
}
```

Output Test 3

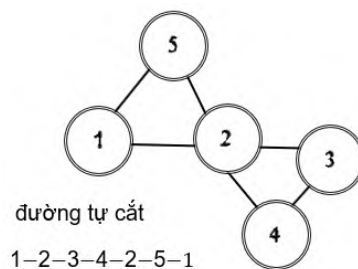
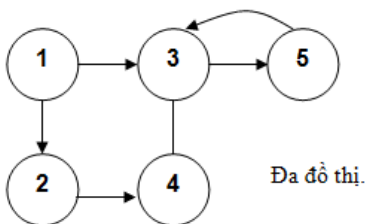
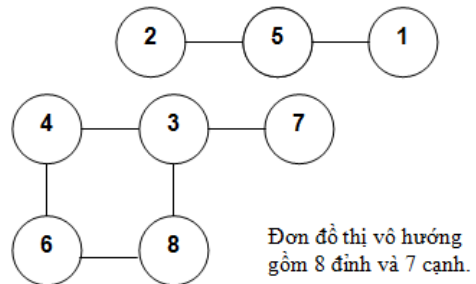
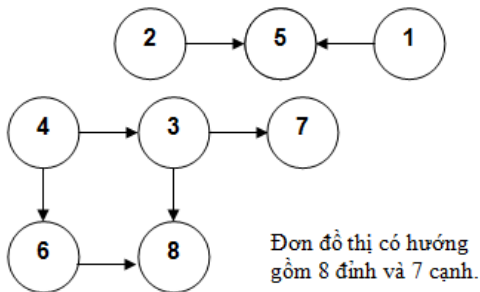
```
Init x = 1/2
F(x++) = 2
now x = 3/2
F(++x) = 10
now x = 5/2
reset x = 1/2
F(x--) = 2
now x = -1/2
F(--x) = -6
now x = -3/2
T h e   E n d
```

Chương 2

GRAPH

Định nghĩa

Một *đồ thị hữu hạn* $G = (V, E)$ bao gồm tập V gồm n đỉnh mã số từ $1..n$ và tập cạnh E với các cặp đỉnh (x, y) nối hai đỉnh x và y . Nếu các cạnh (x, y) đều có chiều đi từ đỉnh x đến đỉnh y , $x \rightarrow y$ thì *đồ thị là có hướng*; ngược lại, khi không quy định hướng đi thì ta thu được *đồ thị vô hướng*. Mỗi cạnh (x, y) của đồ thị vô hướng cho phép ta đi từ đỉnh x đến đỉnh y , $x \rightarrow y$, hoặc ngược lại, từ đỉnh y đến đỉnh x , $x \leftarrow y$ giống như quy định trong giao thông về đường hai chiều. Trong đồ thị có hướng ta dùng thuật ngữ *cung (x, y) thay cho *cạnh* và hiểu là đường một chiều. Trong *đơn đồ thị*, người ta quy định giữa hai đỉnh bất kỳ x và y có không quá 1 cạnh (cung).*



Một *đường p* trong đồ thị là dãy các cạnh nối đầu nhau tạo thành dãy đỉnh $p = (p_1, p_2, \dots, p_k)$, trong đó p_1 là *đỉnh xuất phát*, p_k là *đỉnh kết thúc*, k là *chiều dài* của đường p . Nếu có hai đỉnh trong p trùng nhau thì đường đi gọi là *tự cắt*, ngược lại thì ta có đường *đơn*. Trong đồ thị vô hướng ta luôn coi hai đỉnh kề nhau trên đường p là khác nhau. Tùy theo nội dung của từng bài toán tin, người ta quy định là *có* hay *không có* đường đi từ đỉnh x đến chính nó. Nếu đỉnh cuối của đường trùng với điểm đầu của đường thì ta có một *chu trình*.

Trong phần này, nếu không ghi chú gì thêm, ta ngầm hiểu các đồ thị được cho là *đơn đồ thị hữu hạn và vô hướng*. Cụ thể ta quy định:

- ☞ Đồ thị có $n \geq 0$ đỉnh mã số $1..n$
- ☞ Đồ thị có $m \geq 0$ cạnh (x, y) , $x \neq y$
- ☞ Giữa hai đỉnh x và y có tối đa một cạnh. Không có cạnh (x, x)
- ☞ Nếu có đường đi từ đỉnh x đến đỉnh y thì đó là đường hai chiều, tức là có đường đi từ đỉnh y đến đỉnh x cùng chiều dài.

Đồ thị là công cụ hữu hiệu được sử dụng để biểu diễn và xử lý nhiều bài toán tin.

Find-Union

Tìm-Gộp là kỹ thuật dùng để quản lý hợp của các tập rời nhau. Ta minh họa kỹ thuật này qua bài toán sau:

Park

Cô giáo dẫn các em nhỏ đi chơi công viên. Lớp có n em mã số lần lượt từ 1 đến n . Vào công viên, để các bạn nhỏ khỏi lạc nhau, cô giáo quy định chia nhóm như sau:

Cô lần lượt đọc mã số của từng cặp hai bạn. Hai bạn đó sẽ thuộc cùng nhóm. Dĩ nhiên, nếu bạn A và B cùng nhóm, bạn B và C cùng nhóm thì cả ba bạn A , B và C là cùng nhóm. Cô cũng quy định rằng bạn nào có số hiệu nhỏ nhất trong nhóm sẽ làm nhóm trưởng. Sau m lần ghép cặp như vậy thì các em được chia thành mấy nhóm, hãy liệt kê mã số của các bạn trong từng nhóm.

Input: text file PARK.INP

- ☞ Dòng đầu tiên: hai số nguyên dương n và m ,
- ☞ Dòng thứ i trong số m dòng tiếp theo: mỗi dòng hai số nguyên dương A B là mã số của hai bạn được ghép vào cùng nhóm.

Output: Hiển thị trên màn hình

- ☞ Dòng đầu tiên: k – số lượng nhóm
- ☞ Tiếp đến là k dòng, mỗi dòng liệt kê danh sách một nhóm.

Ví dụ

PARK . INP	OUTPUT
8 5	3
2 5	1 2 5
3 7	3 4 7
8 6	6 8
5 1	
4 3	

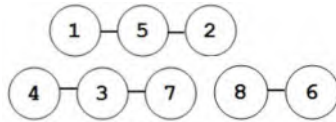
$n = 8$ em nhỏ, 5 lần ghép cặp

Em số 2 ghép cặp với em số 5

Em số 3 ghép cặp với em số 7. . .

...

3 nhóm.



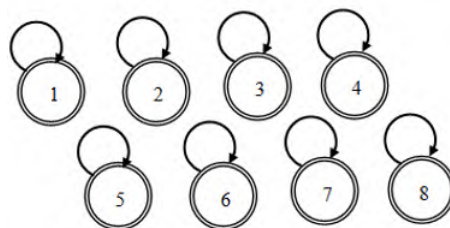
Algorithm

Điều lý thú trong kỹ thuật này là ở điểm nhóm trưởng chính là em mang số hiệu nhỏ nhất trong nhóm. Lúc đầu, chưa ghép cặp thì mỗi em tạo thành một nhóm riêng. Như vậy, lúc đầu ta có n nhóm. Vì mỗi nhóm chỉ có duy nhất 1 em nên chính em đó là nhóm trưởng của chính mình. Ta mượn tượng mỗi bạn x phải bám vai bạn đứng trước y trong cùng nhóm. Theo quy định của cô giáo thì $y < x$. Riêng nhóm trưởng, do có số hiệu nhỏ nhất nên tự bám vai mình.

Ta sử dụng mảng nguyên một chiều $d[1..n]$ để quản lý các nhóm. Ta gán $d[y] = x$ nếu bạn y phải bám vai bạn x là bạn có số hiệu nhỏ hơn mình: $x \leftarrow y$ nếu $y < x$. Lúc đầu ta khởi trị $d[i] = i$; $i = 1..n$: $i \leftarrow i$, với ý nghĩa: lúc đầu, mỗi bạn tạo thành một nhóm 1 người và tự mình làm nhóm trưởng, do đó mỗi bạn tự bám vai chính mình. Mảng d được gọi là *mảng tham chiếu* hoặc *mảng trở ngược*. Như ta sẽ thấy sau này, nhờ d ta có thể nhanh chóng tìm được nhóm trưởng của bất kỳ em nào đồng thời liệt kê được các thành viên trong mỗi nhóm.

Khởi trị

```
for(iny i = 1; i <= n; ++i) d[i] = i;
```



Khởi trị: mỗi em là một nhóm.

d	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]
	1	2	3	4	5	6	7	8

90 CHUYÊN TIN 11: CÁC CHUYÊN ĐỀ CHỌN LỌC VỚI NGÔN NGỮ C++

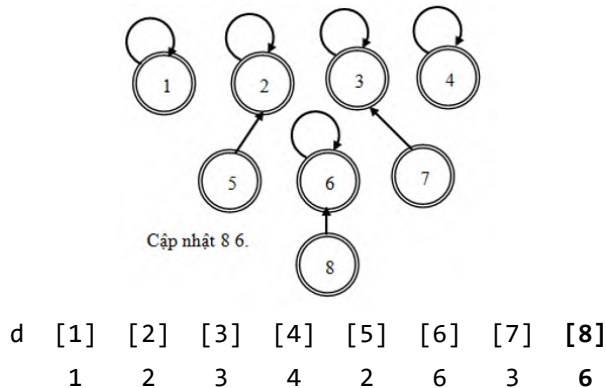
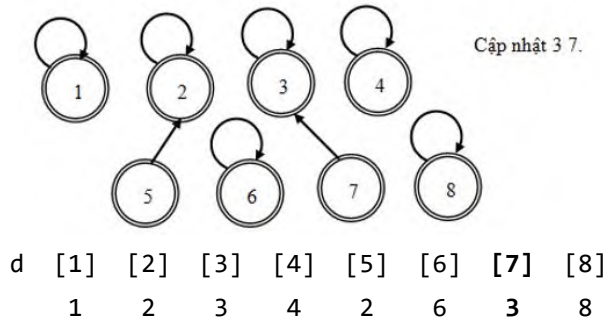
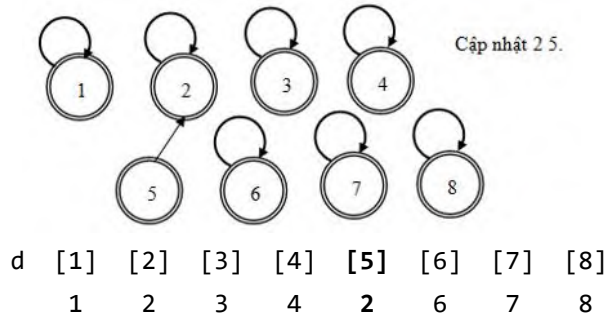
Mỗi khi cô giáo yêu cầu ghép cặp (x, y) thì ta hiểu là: nhóm có em x cần hợp (gộp) với nhóm có em y . Để ghép cặp (x, y) ta thực hiện các bước sau đây:

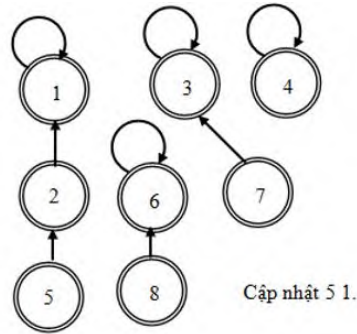
Bước 1. Tìm nhóm trưởng x' của nhóm có em x

Bước 2. Tìm nhóm trưởng y' của nhóm có em y

Bước 3. Quyết định xem ai sẽ là nhóm trưởng của nhóm gộp mới này. Dễ hiểu là nhóm trưởng mới sẽ là em có số hiệu nhỏ nhất trong 2 nhóm trưởng x' và y' , tức là $\min(x', y')$. Sau đó ta gộp 2 nhóm theo kỹ thuật tham chiếu: Giả sử $x' < y'$. Ta chỉ việc gán $d[y'] = x'$: $y' \rightarrow x'$. Ý nghĩa của việc này là: cô giáo quy định nhóm trưởng y' phải bám vai nhóm trưởng x' . Ta suy ra ngay rằng mọi em trong nhóm y' đi theo nhóm trưởng y' , mà y' lại đi theo x' nên mọi em trong nhóm y' sẽ đi theo x' . Nếu $x' > y'$ ta gán $d[x'] = y'$: $x' \rightarrow y'$ với ý nghĩa tương tự như trên. Nếu $x' = y'$ thì có nghĩa là x và y hiện ở trong cùng một nhóm (vì có cùng nhóm trưởng) nên ta không phải làm gì.

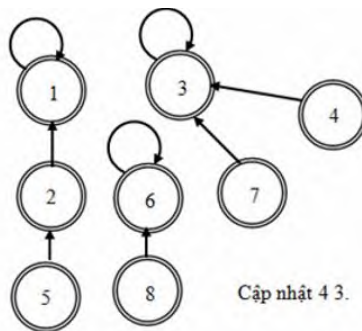
Bạn quan sát lần lượt các hình dưới đây để phát hiện ra quy luật cập nhật từng cặp học sinh theo yêu cầu của cô giáo.





d	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]
	1	1	3	4	2	6	3	6

Để cập nhật cặp (5,1) ta để ý rằng nhóm trưởng của 5 là 2, vì $d[5] = 2$, nhóm trưởng của 1 là 1, vì $d[1] = 1$. Vì $1 < 2$ nên ta cho nhóm trưởng 2 bám vào nhóm trưởng 1, tức là gán $d[2] = 1$: $2 \rightarrow 1$.



d	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]
	1	1	3	3	2	6	3	6

Để ý rằng x sẽ là nhóm trưởng khi và chỉ khi $d[x] = x$. Từ đây ta cài đặt ngay được hàm $\text{Find}(x)$: xác định nhóm trưởng của nhóm có em x như sau:

```
int Find(int x) { // tìm nhóm trưởng của x
    while (d[x] != x) x = d[x];
    return x;
}
```

Để gộp 2 nhóm: nhóm có em x và nhóm có em y ta lưu ý hai điểm sau đây:

- ✂ Rất có thể 2 em x và y hiện đã trong cùng một nhóm. Trường hợp này ta không phải làm gì. Hàm sẽ cho ra giá trị 0. Ta thấy x và y cùng nhóm khi và chỉ khi x và y có cùng nhóm trưởng, tức là $\text{Find}(x) = \text{Find}(y)$.
- ✂ Nếu 2 em x và y thuộc 2 nhóm khác nhau, $\text{Find}(x) \neq \text{Find}(y)$, thì ta thực sự gộp 2 nhóm này và hàm Union cho ra giá trị 1.

92 CHUYÊN TIN 11: CÁC CHUYÊN ĐỀ CHỌN LỌC VỚI NGÔN NGỮ C++

```
// Gộp nhóm chưa x với nhóm chưa y
int Union(int x, int y) {
    x = Find(x); y = Find(y);
    if (x == y) return 0;
    if (x < y) d[y] = x; else d[x] = y;
    return 1;
}
```

Việc cần làm cuối cùng là liệt kê thành viên của mỗi nhóm. Ta duyệt lại mảng d , mỗi khi tìm được một nhóm trưởng i theo hệ thức $d[i] = i$, do i là nhóm trưởng nên mọi thành viên của nhóm này đều có số hiệu bằng hoặc lớn hơn i . Do đặc điểm này nên ta chỉ cần duyệt tiếp mảng d kể từ $j = i$ đến n , nếu nhóm trưởng của j là i thì ta hiển thị thành viên j .

Nhớ rằng lúc đầu ta có tổng cộng n nhóm. Mỗi khi ta gộp thực sự 2 nhóm, tức là khi $\text{Union}(x, y) = 1$ thì số lượng nhóm sẽ giảm đi 1. Nhờ nhận xét này ta dễ dàng tính được số nhóm tại thời điểm kết thúc.

Kết quả của thuật toán sử dụng kỹ thuật Find-Union không phụ thuộc vào trật tự duyệt các cặp ghép.

Độ phức tạp tính toán

Hàm Find thực hiện tối đa n bước tham chiếu $x = d[x]$. Hàm Union gọi hàm Find do đó có độ phức tạp $O(n)$. Tổng cộng lại, với đồ thị n đỉnh, m cạnh ta có độ phức tạp $O(nm)$: tuyến tính theo chiều dài input.

Ghi nhớ

Để vận dụng kỹ thuật Find-Union ta cần:

1. Init: Khai báo một mảng hoặc vector nguyên d , khởi trị $d[i] = i$ với $i = 1..n$
2. Cài đặt hàm Find
3. Cài đặt hàm Union.

4. Sau khi thực hiện xong lượt ghép cặp cuối cùng, bạn nên chuyển mảng tham chiếu d sang dạng tham chiếu trực tiếp. Cụ thể là chuyển đổi:

d	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]
	1	1	3	3	2	6	3	6

thành dạng sau: Thay $d[5] = 2$ (trỏ gián tiếp) thành $d[5] = 1$ (trỏ trực tiếp)

d	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]
	1	1	3	3	1	6	3	6

```
// Dat lai tro truc tiep
for(int i = 2; i <= n; ++i) d[i] = Find(i);
```

Program

```

// Park.CPP: Find-Union
#include<bits/stdc++.h>
using namespace std;
const char * FN = "PARK.INP";
typedef vector<int> VI;
VI d;
int Find(int x) {
    while (d[x] != x) x = d[x];
    return x;
}
int Union(int x, int y) {
    x = Find(x); y = Find(y);
    if (x == y) return 0;
    if (x < y) d[y] = x; else d[x] = y;
    return 1;
}
void Park() {
    ifstream f(FN);
    int n, m;
    f >> n >> m;
    cout << n << " dinh, " << m << " canh" << endl;
    // Khoi tri d
    d.clear();
    for(int i = 0; i <= n; ++i) d.push_back(i);
    int k = n; // so TPLT
    int x, y;
    for(int i = 1; i <= m; ++i) {
        f >> x >> y;
        if (x == y) continue;
        k -= Union(x,y);
    }
    f.close();
    // Tro truc tiep
    for(int i = 2; i <= n; ++i) d[i] = Find(i);
    cout << k << " nhom:";
    for (int i = 1; i <= n; ++i)

```

94 CHUYÊN TIN 11: CÁC CHUYÊN ĐỀ CHỌN LỌC VỚI NGÔN NGỮ C++

```
    if (d[i] == i) {
        cout << endl;
        for (int j = i; j <= n; ++j)
            if (d[j] == i) cout << j << " ";
    }
}

int main() {
    Park();
    cout << "\n T h e   E n d";
    return 0;
}
```

Output

```
Input: [8, 5, 2, 5, 3, 7, 8, 6, 5, 1, 4, 3]
8 đỉnh, 5 cạnh
Init d: [0, 1, 2, 3, 4, 5, 6, 7, 8]
3 nhóm
* Nhóm 1: 1 2 5
* Nhóm 2: 3 4 7
* Nhóm 3: 6 8
T h e   E n d
```

Cách sinh test

✂ Vẽ hình k nhóm

✂ Tạo liên kết cho mỗi nhóm

Ví dụ: xem bài dưới (Connected Zones)

Connected Zones

Cho đồ thị $G = (V, E)$ với n đỉnh $V = \{1..n\}$ và m cạnh (x,y) nối đỉnh x với đỉnh y . Hãy tính số thành phần liên thông (TPLT) và liệt kê tập đỉnh trong mỗi TPLT của G .

Understanding

Đây chính là nội dung của bài Park được trình bày qua ngôn ngữ đồ thị. Đồ thị G được gọi là *liên thông* nếu từ một đỉnh bất kỳ i ta có thể đi theo một số cạnh liền kề nhau của đồ thị để đến một đỉnh j bất kỳ. Dãy các cạnh liền kề từ đỉnh i đến đỉnh j được gọi là một *đường* từ i đến j . Với đồ thị vô hướng, khi có đường từ đỉnh i đến đỉnh j thì ta cũng có đường từ đỉnh j đến đỉnh i . Nếu G không liên thông thì G được chia thành các *mảnh liên thông*, còn gọi là *thành phần liên thông (TPLT)*, gồm một số đỉnh và cạnh của G . Hãy tưởng tượng mỗi đỉnh của đồ thị như là một hạt cườm, cạnh nối hai đỉnh i và j chính

là sợi dây nối hai hạt i và j . Khi đó, G là *liên thông* khi và chỉ khi ta cầm một hạt bất kỳ nhắc lên thì tất cả các hạt đều được nhắc theo. Nếu G không liên thông thì mỗi lần nhắc một hạt ta được một TPLT gồm một số hạt được liên kết với nhau. Ta đặt riêng mảnh đó ra, rồi nhắc một hạt bất kỳ trong số các hạt còn lại ta sẽ được TPLT thứ hai,...

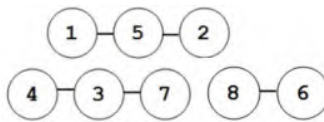
Nhận xét

G là liên thông khi và chỉ khi số TPLT của G là 1.

Đồ thị trong ví dụ dưới đây gồm 3 TPLT $\{1, 5, 2\}$, $\{3, 4, 7\}$ và $\{6, 8\}$.

GRAPH. INP	OUTPUT
8 5	3
2 5	1 2 5
3 7	3 4 7
8 6	6 8
5 1	
4 3	

3 TPLT: $\{1, 2, 5\}$, $\{3, 4, 7\}$ và $\{6, 8\}$



Algorithm

Nhận xét

Số TPLT của G = số nhóm trường.

Để tính số TPLT của G ta thực hiện hai bước sau:

Bước 1. Khởi trị $d[i] = i$; $i = 1..n$; gán $k = n$ dùng để đếm số TPLT.

Bước 2. Đọc dần từng cạnh (x,y) và thực hiện toán tử $k -= \text{Union}(x,y)$;

k sẽ là số TPLT của G .

Lưu ý rằng hàm $\text{Union}(x,y)$ cho giá trị 1 nếu TPLT chứa đỉnh x được gộp thực sự với TPLT chứa đỉnh y , ngược lại, $\text{Union}(x,y) = 0$ nếu hai đỉnh x và y hiện đang cùng có mặt trong một TPLT (do đó không cần thiết gộp hai thành phần này). Như vậy ý nghĩa của giá trị ra của hàm $\text{Union}(x,y)$ là giảm số TPLT của G . Dĩ nhiên, sau khi thực hiện hàm $\text{Union}(x,y)$ thì hai đỉnh x và y sẽ thuộc cùng một TPLT.

Mỗi TPLT của G được gọi là một đồ thị con (subgraph) của đồ thị G . Việc làm trên được gọi là *phân hoạch đồ thị* G thành các đồ thị con.

Các thuật ngữ sau đây được hiểu là nói về cùng một khái niệm:

- ☞ Thuật toán Phân hoạch (viết tắt là PH).
- ☞ Thuật toán Find-Union (viết tắt là FU).
- ☞ Thuật toán tìm các TPLT còn gọi là zone.

Độ phức tạp

$O(nm)$ với n là số đỉnh, m là số cạnh của đồ thị

Program

```
// Connected Zones
#include<bits/stdc++.h>
using namespace std;
const char * FN = "GRAPH.INP";
typedef vector<int> VI;
VI d; // tham chieu dinh
int Find(int x) {
    while (d[x] != x) x = d[x];
    return x;
}
int Union(int x, int y) {
    x = Find(x); y = Find(y);
    if (x == y) return 0;
    if (x < y) d[y] = x; else d[x] = y;
    return 1;
}
void FU() {
    ifstream f(FN);
    int n, m;
    f >> n >> m;
    cout << n << " dinh, " << m << " canh" << endl;
    d.resize(n+1);
    for(int i = 0; i <= n; ++i) d[i] = i;
    int x, y;
    int c = n; // count number of zones
    for(int i = 1; i <= m; ++i) {
        f >> x >> y;
        if (x == y) continue; // bo qua canh dau = cuoi
        c -= Union(x,y);
    }
    f.close();
    // Update d
    for(int i = 1; i <= n; ++i) d[i] = Find(i);
    cout << "\n Connected Zones: " << c;
```

```

int k = 0;
for(int i = 1; i <= n; ++i) {
    if (d[i] == i) {
        ++k;
        cout << "\n zone " << k << ":";
        for(int j = i; j <= n; ++j)
            if (d[j] == i) cout << " " << j;
    }
}
}
int main() {
    FU();
    cout << "\n T h e   E n d";
    return 0;
}

```

Output

```

8 đỉnh, 5 cạnh
Connected Zones: 3
zone 1: 1 2 5
zone 2: 3 4 7
zone 3: 6 8
T h e   E n d

```

Ghi nhớ

Sau khi vận dụng kỹ thuật Find-Union mảng tham chiếu d cho ta các thông tin sau đây:

Với mỗi đỉnh $i = 1..n$

- ☞ $d[i] = i$ khi và chỉ khi đỉnh i là "nhóm trưởng", tức là đỉnh đại diện cho một thành phần liên thông trong đồ thị;
- ☞ Số lượng thành phần liên thông trong G = số lượng nhóm trưởng = số lượng các đỉnh i thỏa điều kiện $d[i] = i$;
- ☞ Hai đỉnh i và j thuộc cùng một thành phần liên thông khi và chỉ khi $\text{Find}(i) = \text{Find}(j)$;
- ☞ Mỗi đỉnh i thuộc một thành phần liên thông duy nhất.

Connectivity

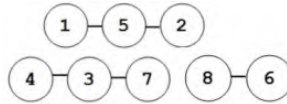
Cho đồ thị vô hướng $G = (V, E)$ với n đỉnh $V = \{1..n\}$ và m cạnh (x,y) nối đỉnh x với đỉnh y . Hãy hiển thị 1 nếu đồ thị G liên thông, 0 nếu G không liên thông.

Ví dụ

GRAPH . INP	OUTPUT
8 5 2 5 3 7 8 6 5 1 4 3	0

8 đỉnh và 5 cạnh.

G không liên thông vì có hơn 1 TPLT (3)



Algorithm

Đồ thị G liên thông khi và chỉ khi số TPLT bằng 1.

Độ phức tạp: $O(nm)$, n là số đỉnh, m là số cạnh của G.

Spanning Tree

Cây khung (còn gọi là *cây bao trùm*) của một đồ thị liên thông vô hướng n đỉnh, m cạnh là cây gồm n đỉnh với số cạnh tối thiểu bảo toàn tính liên thông của đồ thị.

Nhận xét: *Cây khung của đồ thị n đỉnh, m cạnh có đúng n đỉnh và n-1 cạnh.*

Về mặt trực quan, nếu ta tưởng tượng đồ thị như một bộ đèn chùm treo trên trần nhà với mỗi bóng đèn là một đỉnh, mỗi thanh cứng nối hai bóng đèn là một cạnh của đồ thị thì yêu cầu đặt ra là có thể bỏ bớt một số cạnh mà các bóng đèn không rơi xuống đất được không.

Số cạnh tối thiểu còn lại giữ cho các bóng đèn không rơi được gọi là *cây khung*.

Có hai thuật toán, hai cách tiếp cận để tìm cây khung là thuật toán Prim và thuật toán Kruskal.

- ☞ Thuật toán Prim có ý tưởng như vừa trình bày: Treo chùm đèn rồi thử tháo dần từng thanh nối. Nếu tháo thanh nối (x,y) làm cho bóng đèn rơi ra thì lắp lại thanh nối đó. Độ phức tạp $O(m)$, m là số cạnh của đồ thị.
- ☞ Thuật toán Kruskal thao tác ngược lại với thuật toán Prim: Xuất phát từ đèn chùm rỗng, chỉ có một móc treo trên trần, ta lắp dần các thanh nối hai bóng đèn (x,y). Nếu hai bóng này trước đó đã được lắp bởi các thanh nối rồi, tức là x và y đã liên thông với nhau thì bỏ qua thanh nối (x,y). Ta thấy thuật toán Kruskal chính là thuật toán Find - Union được vận dụng trong bài Park và bài Connected Zones.

Algorithm

Duyệt các cạnh: nếu cạnh (x,y) không tạo thành chu trình với cây khung thì nạp (x,y) vào cây khung.

Điều kiện để cạnh (x,y) không tạo thành chu trình với cây khung là:

$\text{Union}(x,y) = 1.$

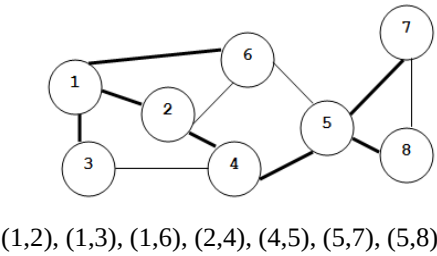
Khi nạp đủ n-1 cạnh thì kết thúc thuật toán.

Độ phức tạp: $O(nm)$.

GRAPH. INP	OUTPUT
8 11	1 2
1 2	1 3
1 3	1 6
1 6	2 4
2 4	4 5
2 6	5 7
3 4	5 8
4 5	
5 6	
5 7	
5 8	
7 8	

Input: Đồ thị có 8 đỉnh và 11 cạnh.

Output: 7 cạnh của cây khung (tô đậm)



Program

Đồ thị hữu hạn có cây khung khi và chỉ khi đồ thị là liên thông. Như vậy, khi số $TPLT > 1$ thì đồ thị không liên thông và do đó không có cây khung.

```
// Spanning Tree (Algorithm Kruskal)
#include<bits/stdc++.h>
using namespace std;
const char * FN = "GRAPH.INP";
typedef vector<int> VI;
VI d; // tham chiếu đỉnh
int Find(int x) {
    while (d[x] != x) x = d[x];
    return x;
}
int Union(int x, int y) {
    x = Find(x); y = Find(y);
```

100 CHUYÊN TIN 11: CÁC CHUYÊN ĐỀ CHỌN LỌC VỚI NGÔN NGỮ C++

```
    if (x == y) return 0;
    if (x < y) d[y] = x;
    else d[x] = y;
    return 1;
}

void ST() { // Spanning Tree
    ifstream f(FN);
    int n, m;
    f >> n >> m;
    cout << n << " vertices, " << m << " edges" << endl;
    d.clear(); // Khoi tri tro dinh d
    for(int i = 0; i <= n; ++i) d.push_back(i);
    int x, y, c = 0; // c dem so canh khung
    cout << " Spanning tree has " << n-1 << " edges: " << endl;
    for(int i = 1; i <= m; ++i) {
        f >> x >> y;
        if (x == y) continue; // bo qua canh dau = cuoi
        if (!Union(x,y)) continue; // bo qua canh tao chu trinh
        cout << x << " " << y << endl;
        ++c;
        if (c == n-1) break;
    }
    f.close();
}

int main() {
    ST();
    cout << "\n T h e   E n d";
    return 0;
}
```

Output

```
8 dinh, 11 canh
Cay khung gom 7 canh:
1 2
1 3
1 6
2 4
```


Program

```

// Minimal Spanning Tree (Algorithm Kruskal)
#include<bits/stdc++.h>
using namespace std;
const char * FN = "GRAPH.INP";
vector<int> d; // tro dinh
int n; // so dinh
class Edge {
public:
    int X, Y; // canh (x,y)
    int W; // trong so
    Edge(int x, int y, int w) { // constructor
        if (x < y) { X = x; Y = y; }
        else { X = y; Y = x; }
        W = w;
    }
    void Print(const char * msg = "") {
        cout << msg << X << " " << Y << ": " << W;
    }
};
typedef vector<Edge> VE;
int Find(int x) {
    while (d[x] != x) x = d[x];
    return x;
}
int Union(int x, int y) {
    x = Find(x); y = Find(y);
    if (x == y) return 0;
    if (x < y) d[y] = x; else d[x] = y;
    return 1;
}
VE Input() {
    cout << "\n Input: ";
    ifstream f(FN);
    if(!f) { cout << "\n unable open input file " << FN; exit(0); }

```

```

int m;
f >> n >> m;
cout << n << " vertices, " << m << " edges" << endl;
VE v;
int x, y, w;
for(int i = 1; i <= m; ++i) {
    f >> x >> y >> w;
    if (x == y) continue;
    Edge e(x,y,w);
    v.push_back(e);
    e.Print("\n ");
}
f.close();
return v;
}

inline bool Less(Edge a, Edge b) { return a.W < b.W; }

void MST() {
    VE v = Input();
    sort(v.begin(), v.end(), Less);
    cout << "\n Sorted e: " << endl;
    for(Edge e:v) e.Print("\n");
    // Tao cay khung
    d.clear(); // Khoi tri tro dinh d
    for(int i = 0; i <= n; ++i) d.push_back(i);
    int c = 0; // count canh khung
    cout << "\n Min spanning tree has " << n-1 << " edges: " << endl;
    int wsum = 0; // sum of weights
    for(Edge e:v) {
        if (Union(e.X, e.Y)) {
            e.Print("\n");
            wsum += e.W;
            ++c; if (c == n-1) break;
        }
    }
    cout << "\n Weight sum: " << wsum;
}

```

104 CHUYÊN TIN 11: CÁC CHUYÊN ĐỀ CHỌN LỌC VỚI NGÔN NGỮ C++

```
int main() {  
    MST();  
    cout << "\n T h e    E n d";  
    return 0;  
}
```

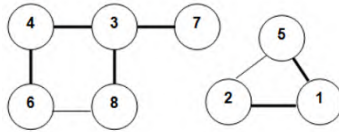
Output

```
Input: 8 vertices, 11 edges  
1 2: 2  
.  
.  
.  
Sorted e:  
1 2: 2  
5 8: 2  
7 8: 3  
2 4: 4  
5 7: 4  
5 6: 5  
3 4: 6  
1 3: 7  
2 6: 8  
1 6: 9  
4 5: 10  
Min spanning tree has 7 edges:  
1 2: 2  
5 8: 2  
7 8: 3  
2 4: 4  
5 6: 5  
3 4: 6  
2 6: 8  
Weight sum: 30  
T h e    E n d
```

Spanning Forest

Cho G là một đồ thị vô hướng gồm n đỉnh và m cạnh. Tập hợp các cây khung trong mỗi mảnh liên thông của G được gọi là rừng khung của G . Hãy liệt kê rừng khung của G .

GRAPH. INP	Đồ thị có 8 đỉnh và 8 cạnh.
8 8	Rừng khung gồm hai cây khung (tô đậm)
1 2	Cây khung liên thông thuộc đỉnh 1 gồm 2 cạnh:
3 4	(1,2), (1,5).
3 7	Cây khung liên thông thuộc đỉnh 3 gồm 4 cạnh:
1 5	(3,4), (3,7), (3,8), (4,6).
3 8	
4 6	
6 8	
2 5	



Algorithm

Để ý rằng dù đồ thị (vô hướng) liên thông hay không liên thông thì thủ tục ST như mô tả trong các bài trước đều cho ta một danh sách các cạnh được chọn vào cây khung cần tìm. Danh sách này được ghi trong vector $e \langle \text{Edge} \rangle$, với Edge là kiểu mô tả cạnh của đồ thị gồm hai trường x, y .

Ta quy ước biểu diễn đồ thị $G = (V, E)$ ban đầu như sau:

G có N đỉnh mã số từ $1..N$: $V = \{1, 2, \dots, N\}$

G có M cạnh $E = \{(x,y), x < y, x,y \in V\}$

Ba giá trị N, M và E là cố định cho mỗi bài.

Khi cần nói đến các tập con của tập cạnh E ta dùng các chỉ số $0..M-1$. Chỉ số i cho ta cạnh $E[i]$.

Thuật toán Rừng khung SF gồm các bước sau đây:

- ☞ Đọc dữ liệu N, M và các cạnh E . Độ phức tạp $O(m)$
- ☞ Dùng thuật toán Find-Union (FU) trên G để xác định các miền liên thông (zones). Thông tin ra sẽ là vector $d[0..N+1]$ thể hiện mối quan hệ giữa các đỉnh. Đỉnh i là head (nhóm trưởng) khi và chỉ khi $d[i] = i$. Ta cũng quy ước $d[0]$ chứa số lượng zone. Độ phức tạp $O(nm)$
- ☞ Với mỗi zone (TPLT) ta xác định, thông qua d , tập z gồm các đỉnh thuộc zone đó. Gọi thuật toán $ST(z)$ để tìm cây khung (spanning tree) của zone z . Độ phức tạp $O(nm)$
- ☞ Algorithm $ST(z)$: Duyệt các cạnh $e: E$ có hai đỉnh nằm hoàn toàn trong z . Nếu $\text{Union}(e.X, e.Y)$ thì xuất ra kết quả.

Từ bài này trở đi, ta bổ sung tham biến d cho các hàm Find và Union để có thể áp dụng cho các đồ thị con của G .

Nhận xét

Tổng số cạnh trong các cây khung của rừng khung là số đỉnh – số TPLT.

Thật vậy, giả sử đồ thị có n đỉnh và k TPLT. Gọi $N(i)$ là số đỉnh trong mảnh liên thông thứ i ; $i = 1..k$, T là tổng số cạnh trong các cây của rừng khung. Để ý rằng n đỉnh được phân hoạch vào k mảnh liên thông (không giao nhau) và cây khung của mảnh liên thông thứ i có $N(i)-1$ cạnh; $i = 1..k$. Ta có:

$$N(1) + N(2) + \dots + N(k) = n$$

$$T = (N(1)-1) + (N(2)-1) + \dots + (N(k)-1) = n - k.$$

Program

```
// Spanning Forest
#include<bits/stdc++.h>
using namespace std;
const char * FN = "GRAPH.INP";
typedef vector<int > VI;
int N; // number of vertices
int M; // number of edges
class Edge {
public:
    int X, Y; // canh (x,y)
    // constructor
    Edge(int x, int y) {
        if(x < y) { X = x; Y = y; }
        else { X = y; Y = x; }
    }
    void Print(const char * msg = "") {cout << msg << "("<<X<< ", "<< Y<< ")"; }
};
typedef vector<Edge> VE;
VE E; // edges
// init and return pointer of vertices d
VI Init(int n) {
    VI d(n+1);
    for(int i = 0; i <= n; ++i) d[i] = i;
    return d;
}
// head of set containing vertex x. O(n)
int Find(const VI &d, int x) {
```

```

    while (d[x] != x) x = d[x];
    return x;
}
// Union two subsets containing x and y. O(n)
int Union(VI & d, int x, int y) {
    x = Find(d, x); y = Find(d, y);
    if (x == y) return 0;
    if (x < y) d[y] = x; else d[x] = y;
    return 1;
}
// change d[i] = find(i), compute d[0] = number of zones
// return d[0], O(n^2)
int Update(VI & d) {
    for(int i = 1; i <= N; ++i)
        d[i] = Find(d, i);
    for(int i = 1; i <= N; ++i) if (d[i] == i) ++d[0];
    return d[0];
}
// Find-Union
VI FU() {
    VI d = Init(N);
    for(auto e:E) Union(d, e.X, e.Y);
    Update(d);
    return d;
}
void Input() {
    cout << " Input: " << endl;
    ifstream f(FN);
    if(!f) { cout << "\n unable open input file " << FN; exit(0); }
    f >> N >> M;
    cout << N << " vertices, " << M << " edges" << endl;
    E.clear();
    int x, y;
    for(int i = 1; i <= M; ++i) {
        f >> x >> y;
        Edge e(x,y);
        E.push_back(e);
    }
}

```

```

        e.Print("\n ");
    }
    f.close();
}
// Edge e in vertices z ?
bool IsIn(const VI &z, Edge e) {
    return count(z.begin(), z.end(), e.X)
        && count(z.begin(), z.end(), e.Y);
}
// spanning tree of zone z
VE ST(const VI &z) {
    VE st;
    VI d = Init(N);
    for(auto e:E)
        if(IsIn(z, e)) {
            if (Union(d, e.X, e.Y))
                st.push_back(e);
        }
    return st;
}
// spanning forest
void SF() {
    Input();
    VI d = FU();
    cout << "\n number of spanning trees: " << d[0];
    int c = 0; // counter
    for(int i = 1; i <= N; ++i) {
        if (d[i] == i) { // i is head
            VI zone; // vertices in zone i
            for(int j = i; j <= N; ++j)
                if (d[j] == i) zone.push_back(j);
            ++c;
            cout << "\n spanning tree No " << c << ": ";
            for(auto x:ST(zone)) x.Print(" ");
        }
    }
}

```

```

int main() {
    SF();
    cout << "\n T h e   E n d";
    return 0;
}

```

Output

```

Input:
8 vertices, 8 edges
(1, 2)
(3, 4)
(3, 7)
(1, 5)
(3, 8)
(4, 6)
(6, 8)
(2, 5)
number of spanning trees: 2
spanning tree No 1: (1, 2) (1, 5)
spanning tree No 2: (3, 4) (3, 7) (3, 8) (4, 6)
T h e   E n d

```

Improve

Chương trình trên phải thực hiện Find-Union một số lần, lần đầu tiên trên toàn đồ thị, các lần khác trong mỗi zone, do đó việc tổ chức dữ liệu có khó khăn. Chúng ta cải tiến như sau. Dùng một vector mark đánh dấu các chỉ số cạnh, cạnh nằm trong khung sẽ được đánh dấu 1 tương ứng với lời gọi Union thành công, cạnh tạo chu trình, Union = 0 sẽ được đánh dấu 0. Sau khi thực hiện Find-Union trên đồ thị ta nhận được các vector d và mark. Từ hai vector này ta xác định được cây khung cho mỗi mảnh liên thông.

Program

```

/*-----
Spanning Forest
Improve: mark edge: 2 in spanning: 1 not in spanning: 0
-----*/

#include<bits/stdc++.h>
using namespace std;
const char * FN = "GRAPH.INP";
typedef vector<int > VI;

```

110 CHUYÊN TIN 11: CÁC CHUYÊN ĐỀ CHỌN LỌC VỚI NGÔN NGỮ C++

```

int N; // number of vertices
int M; // number of edges
VI d; // tro dinh
VI mark; // canh 2, khung 1, tao chu trinh 0
class Edge {
public:
    int X, Y; // canh (x,y)
    // constructor
    Edge(int x, int y) {
        if(x < y) { X = x; Y = y; }
        else { X = y; Y = x; }
    }
    void Print(const char * msg = "") { cout << msg << "(" << X << ", " << Y << ")"; }
};
typedef vector<Edge> VE;
VE E; // edges
// init pointer to vertices d
void Init() { for(int i = 0; i <= N; ++i) d[i] = i; }
// head of set containing vertex x. O(n)
int Find(int x) {
    while (d[x] != x) x = d[x];
    return x;
}
// Union two subsets containing x and y. O(n)
int Union(int x, int y) {
    x = Find(x); y = Find(y);
    if (x == y) return 0;
    if (x < y) d[y] = x; else d[x] = y;
    return 1;
}
int Union(Edge e) { return Union(e.X, e.Y); }
// change d[i] = find(i). compute d[0] = number of zones
// return d[0] O(n^2)
int Update() {
    for(int i = 1; i <= N; ++i) d[i] = Find(i);
    for(int i = 1; i <= N; ++i) if (d[i] == i) ++d[0];
    return d[0];
}

```

```

void Input() {
    cout << " Input: " << endl;
    ifstream f(FN);
    if(!f) { cout << "\n unable open input file " << FN;  exit(0);  }
    f >> N >> M;
    cout << N << " vertices, " << M << " edges" << endl;
    // read Edges
    E.clear();
    int x, y;
    for(int i = 1; i <= M; ++i) {
        f >> x >> y;
        Edge e(x,y);
        E.push_back(e);
        e.Print("\n ");
    }
    f.close();
}

// Edge e in vertices z ?
bool IsIn(const VI &z, Edge e) {
    return count(z.begin(), z.end(), e.X)
        && count(z.begin(), z.end(), e.Y);
}

// spanning tree of zone z
VE ST(const VI &z) {
    VE v;
    // duyệt lại các cạnh trong e
    // cạnh nào có 2 đỉnh nằm trong z
    // và mark = 1 thì đưa vào kết quả v
    for(int i = 0; i < M; ++i) {
        if(mark[i] && IsIn(z, E[i]))
            v.push_back(E[i]);
    }
    return v;
}

// spanning forest
void SF() {
    Input();

```

```

    d.assign(N+1, 0);
    Init();
    mark.assign(M, 2); // canh thuong
    for(int i = 0; i < M; ++i) mark[i] = Union(E[i]);
    cout << "\n Edge mark: ";
    for(int i = 0; i < M; ++i) {
        cout << "\n " << mark[i];
        E[i].Print(" : ");
    }
    cout << "\n number of spanning trees: " << d[0];
    int c = 0; // counter
    for(int i = 1; i <= N; ++i) {
        if (d[i] == i) { // i is head
            VI zone; // vertices in zone i
            for(int j = i; j <= N; ++j)
                if (d[j] == i) zone.push_back(j);
            ++c;
            cout << "\n spanning tree No " << c << ": ";
            for(auto x:ST(zone)) x.Print(" ");
        }
    }
}

int main() {
    SF();
    cout << "\n T h e   E n d";
    return 0;
}

```

Output

```

Input:
8 vertices, 8 edges
. . .
Edge mark:
1 : (1, 2)
1 : (3, 4)
1 : (3, 7)
1 : (1, 5)
1 : (3, 8)

```

```

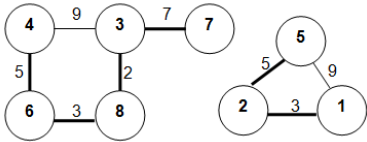
1 : (4, 6)
0 : (6, 8)
0 : (2, 5)
number of spanning Trees: 0
spanning tree No 1: (1, 2) (1, 5)
spanning tree No 2: (3, 4) (3, 7) (3, 8) (4, 6)
T h e   E n d

```

Minimal Spanning Forest

Rừng khung cực tiểu được xác định như rừng khung với phép duyệt các cạnh theo trật tự tăng dần của trọng số cạnh.

Mỗi cạnh trong bài này được mô tả như trong bài Cây khung cực tiểu: $e = \{(x,y):w\}$, trong đó (x,y) là cạnh, w là trọng số của cạnh.

graph.inp	Đồ thị có 8 đỉnh và 8 cạnh dạng xyp .
8 8	Rừng gồm hai cây khung (tô đậm)
1 2 3	
1 5 9	
2 5 5	
3 4 9	
3 7 7	
3 8 2	
4 6 5	
6 8 3	
	Cây khung liên thuộc đỉnh 1 gồm 2 cạnh: $(1,2), (2,5)$ và trọng số $= 3+5 = 8$.
	Cây khung liên thuộc đỉnh 3 gồm 4 cạnh: $(3,7), (3,8), (4,6), (6,8)$ và trọng số $= 7+2+5+3 = 17$.
	Tổng trọng số min $= 8+17 = 25$.

Algorithm

Thuật toán hoạt động giống như Spanning Forest ngoại trừ pha đầu tiên: sắp tăng các cạnh theo trọng số w .

☞ Đọc dữ liệu

☞ Sắp tăng E theo trọng số w

☞ Spanning Forest

```

inline bool LessW(Edge a, Edge b) {
    return a.W < b.W;
}

// minimal spanning forest
void MSF() {
    Input();

```


114 CHUYÊN TIN 11: CÁC CHUYÊN ĐỀ CHỌN LỌC VỚI NGÔN NGỮ C++

```
sort(E.begin(), E.end(), LessW);
d.assign(N+1, 0);
Init();
mark.assign(M, 2); // canh thuong
for(int i = 0; i < M; ++i) mark[i] = Union(E[i]);
cout << "\n Edge mark: ";
for(int i = 0; i < M; ++i) {
    cout << "\n " << mark[i];
    E[i].Print(" : ");
}
cout << "\n number of spanning trees: " << d[0];
int c = 0; // counter
int wsum = 0;
for(int i = 1; i <= N; ++i) {
    if (d[i] == i) { // i is head
        VI zone; // vertices in zone i
        for(int j = i; j <= N; ++j)
            if (d[j] == i) zone.push_back(j);
        ++c;
        cout << "\n spanning tree No " << c << ": ";
        int ws = 0;
        for(auto x:MST(zone)) {
            x.Print(" ");
            ws += x.W;
        }
        cout << "\n partial sum = " << ws;
        wsum += ws;
    }
}
cout << "\n Total sum = " << wsum;
}
```

Cycle

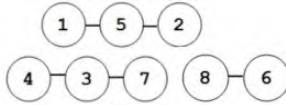
Cho đồ thị vô hướng $G = (V, E)$ với n đỉnh $V = \{1..n\}$ và m cạnh (x, y) nối đỉnh x với đỉnh y . Hãy hiển thị 1 nếu đồ thị G có chứa ít nhất một chu trình, 0 nếu G không có chu trình.

Ví dụ 1

graph.inp	Output
8 5	0
2 5	
3 7	
8 6	
5 1	
4 3	

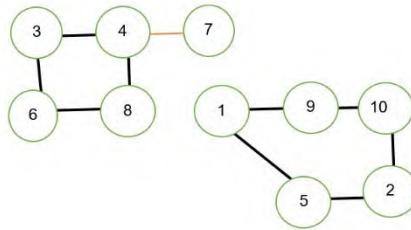
Đồ thị có 8 đỉnh và 5 cạnh.

Đồ thị không có chu trình: 0.



Ví dụ 2

graph.inp	Output
10 10	1
9 1	
2 5	
4 7	
8 6	
5 1	
4 3	
9 10	
10 2	
4 8	
3 6	



Đồ thị có 2 chu trình

Algorithm

Nhận xét 1. Đồ thị vô hướng G có chu trình khi và chỉ khi trong quá trình ghép cạnh ta gặp một cạnh (x, y) thỏa tính chất $x \neq y$ và $\text{Union}(x, y) = 0$.

Thật vậy, $\text{Union}(x, y) = 0$ chứng tỏ tại thời điểm trước khi thực hiện hàm $\text{Union}(x, y)$, hai đỉnh x và y thuộc cùng một TPLT, nghĩa là có đường đi từ x đến y . Nay ta thêm cạnh (x, y) tức là ta có thể đi tiếp thêm một bước nữa từ x đến y , hoặc từ y đến x vì đồ thị là vô hướng. Như vậy là đã xuất hiện một chu trình từ x trở về x . Ta gọi TPLT này là *TPLT khép kín*.

Nhận xét 2. Nếu trong đồ thị $G = (V, E)$, số đỉnh $\leq$ số cạnh (khác nhau), cụ thể là nếu $\#V \leq \#E$ thì đồ thị có chu trình ($\#$ là ký hiệu số lượng).

Thật vậy, ta biết nếu đồ thị hữu hạn, vô hướng thì số cạnh trong các cây khung là $C = N - K$, trong đó N là số đỉnh của đồ thị, K là số thành phần liên thông. Gọi số cạnh khác nhau của đồ thị là M . Khi $N \leq M$ thì $C = N - K \leq N \leq M$, suy ra $C < M$. Do đó trong quá trình ghép cạnh ta sẽ gặp tình huống $\text{Union}(x, y) = 0$, tức là đồ thị có chu trình.

Tuy nhiên, nhận xét 2 không phải là *điều kiện cần và đủ*. Bạn vẽ đồ thị với 10 đỉnh nhưng chỉ nối 3 đỉnh 1, 2, 3 thành một tam giác. Bạn có: $N = 10$, $M = 3$, $N > M$, đồ thị này vẫn có chu trình.

Ta có thể sửa lại một chút như sau:

Nhận xét 3. *Nếu trong đồ thị $G = (V, E)$, N là số đỉnh thực dùng, M là số cạnh (khác nhau), thực dùng thì G có chu trình khi và chỉ khi $N \leq M$.*

Nhận xét này đòi hỏi phải kiểm tra các đỉnh và các cạnh thực dùng nên có phần phức tạp hơn nhận xét 1.

Chương trình dưới đây được cài đặt theo nhận xét 1.

Đọc dữ liệu và thực hiện Find-Union FU. Nếu FU = 0 thì return 1 (có chu trình), ngược lại thì return 0 (không có chu trình).

Độ phức tạp: $O(nm)$

Program

```
// Cycle
#include<bits/stdc++.h>
using namespace std;
const char * FN = "GRAPH.INP";
vector<int> d;
int Find(int x) {
    while (d[x] != x) x = d[x];
    return x;
}
int Union(int x, int y) {
    x = Find(x); y = Find(y);
    if (x == y) return 0;
    if (x < y) d[y] = x; else d[x] = y;
    return 1;
}
int Cycle() {
    ifstream f(FN);
    if(!f) { cout << "\n unable open input file " << FN; exit(0); }
    int n, m;
    f >> n >> m;
    for(int i = 0; i <= n; ++i) d.push_back(i);
    int x, y ;
    for(int i = 1; i <= m; ++i) {
```

```

f >> x >> y;
if (x == y) continue; // bỏ qua cạnh đầu = cuối
if (Union(x,y) == 0) { f.close(); return 1; }
}
f.close();
return 0;
}

int main() {
    cout << Cycle();
    cout << "\n T h e   E n d";
    return 0;
}

```

Cycle In Connected Zone

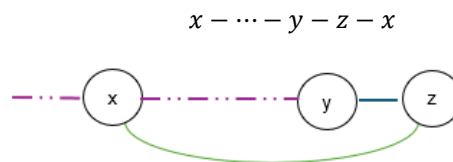
1. Chương trình trên chỉ đơn giản hiển thị 1: đồ thị có chu trình, hoặc 0: không có chu trình.

2. Trong trường hợp đồ thị có chu trình và bạn muốn chỉ ra một trong số các chu trình thì bạn có thể theo thuật toán sau đây:

2.1: Đọc dữ liệu vào ma trận kề a , mỗi cạnh (x,y) sẽ tạo ra hai phép gán trị: $a[x][y] = a[y][x] = 1$

Dùng kỹ thuật Find-Union phát hiện chu trình và TPLT khép kín cz chứa chu trình cuối cùng mà bạn phát hiện được. Ví dụ trên cho ta $cz = [3, 4, 6, 7, 8]$.

2.2: Phỏng theo thuật toán quay lui trong bài Sợi chỉ Arian (trang 248, Chuyên Tin 10) tìm một đường đi v từ đỉnh x đến các đỉnh còn lại thuộc TPLT cz . Nếu đường đi v có $\text{len} \geq 3$, cụ thể là v có dạng $v = [..x..y,z]$ thì kiểm tra xem có tồn tại cạnh (x,z) hay không. Nếu có, thì ta sẽ nhận được chu trình:



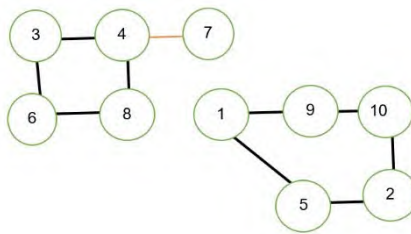
Gọi cz là tập các đỉnh trong TPLT khép kín, ta cần tìm chu trình trong cz , tức là tìm một đường khép kín $vv = x-y\dots x$, $\text{len}(vv) > 3$ chứa các đỉnh trong cz .

☞ Nếu $\text{len}(cz) < 3$ thì cz không chứa chu trình.

- ✧ Nếu $\text{len}(cz) = 3$: thì bản thân cz sẽ tạo thành chu trình sau khi thêm phần tử đầu tiên vào cuối cz .
- ✧ Nếu $\text{len}(cz) > 3$ thì chắc chắn cz chứa một chu trình nào đó, ta tìm chu trình này theo tiếp cận back tracking như sau:
 - Xuất phát từ $x = cz[0]$
 - Tìm đường đi v bắt đầu từ x tới đỉnh $z \neq x$ trong cz . Giả sử $v = [x, y, \dots, z]$
 - Mỗi khi tìm được thêm đỉnh z ta kiểm tra xem từ z trở về trước có đỉnh x nào nối với z hay không. Nếu có đỉnh x thỏa: x xuất hiện trước y , y là đỉnh sát trước z , có cạnh (x, z) và $\text{len}(v) \geq 3$ thì tức là xuất hiện chu trình (xem hình vẽ và hàm Goal).

Ví dụ

graph.inp	Output
10 10	1
9 1	3 4 8 6 3
2 5	
4 7	
8 6	
5 1	
4 3	
9 10	
10 2	
4 8	
3 6	



Đồ thị có 2 chu trình

Chương trình hiển thị chu trình

3 - 4 - 8 - 6 - 3

Cycle in connected zone cz

```
VI GetCycle(VI vv, int i) {
    VI w;
    for(int j = i; j < vv.size(); ++j) w.push_back(vv[j]);
    w.push_back(vv[i]);
    return w;
}

VI Goal(VI v) {
    // xét các đỉnh  $x > 0$  trong path  $v$ 
    VI vv;
    for(int i = 0; i < v.size(); ++i) {
        if (v[i] > 0) vv.push_back(v[i]);
    }
}
```

```

    if (vv.size() < 3) return Empty;
    // len(vv) >= 3: vv = [...z]
    int z = vv.back();
    for(int i = vv.size()-3; i >= 0; --i) // canh (x,z) tao cycle ?
        if (a[vv[i]][z] == 1) return GetCycle(vv, i);
    return Empty;
}

// Tim duong di tu x -> z?
int Next(VI cz, int x) {
    for(int i = 0; i < cz.size(); ++i) {
        int z = cz[i];
        if(z == x) continue;
        if (visit[z]) continue;
        if (a[x][z]) return z;
    }
    return -1;
}

VI Cycle(VI cz) { // Backtracking
    int czlen = cz.size();
    if (czlen == 0) return cz;
    if (czlen == 3) {
        cz.push_back(cz.front());
        return cz;
    }
    visit.assign(n+1, false);
    VI v;
    v.assign(czlen, 0); // path
    int x = cz[0], k = 0;
    v[k] = x; // xuất phát từ đỉnh x
    visit[x] = true; // đã thăm đỉnh x
    while (true) {
        VI vv = Goal(v);
        if (vv.size() > 3) return vv; // den dich
        if (k < 0) return Empty;
        int z = Next(cz, v[k]); // v[k] -> z ?
        if (z > 0) { // tien
            v[++k] = z;

```

```

        visit[z] = 1; // đã thăm đỉnh z
    }
    else --k; // lui
} // while
}

```

Độ phức tạp: $O(n)$

Program

```

// Cycle.CPP: cycle in connected zone cz
#include<bits/stdc++.h>
using namespace std;
typedef vector<int> VI;
typedef vector<char> VC;
typedef vector<bool> VB;
VI d; // tro dinh
VI Empty; // empty vector
int n, m; // n dinh n, m canh
vector<VC> a; // ma tran ke
VB visit;
const char * FN = "GRAPH.INP";
void Print(VI v, const char * msg = "") {
    cout << msg;
    for (int i = 0; i < v.size(); ++i) cout << " " << v[i] ;
}
int Find(int x) {
    while (d[x] != x) x = d[x];
    return x;
}
int Union(int x, int y) {
    x = Find(x); y = Find(y);
    if (x == y) return 0;
    if (x < y) d[y] = x; else d[x] = y;
    return 1;
}
VI GetCycle(VI vv, int i) {
    VI w;
    for(int j = i; j < vv.size(); ++j) w.push_back(vv[j]);
    w.push_back(vv[i]);
}

```

```

    return w;
}
VI Goal(VI v) {
    // xét các đỉnh x > 0 trong path v
    VI vv;
    for(int i = 0; i < v.size(); ++i) {
        if (v[i] > 0) vv.push_back(v[i]);
    }
    if (vv.size() < 3) return Empty;
    // len(vv) >= 3: vv = [...z]
    int z = vv.back();
    for(int i = vv.size()-3; i >= 0; --i) { // canh (x,z) tao cycle ?
        if (a[vv[i]][z]) return GetCycle(vv, i); }
    return Empty;
}
// Tim duong di tu x -> z?
int Next(VI cz, int x) {
    for(int i = 0; i < cz.size(); ++i) {
        int z = cz[i];
        if(z == x) continue;
        if (visit[z]) continue;
        if (a[x][z]) return z;
    }
    return -1;
}
// Cycle trong TPLT cz
VI Cycle(VI cz) { // Backtracking
    int czlen = cz.size();
    if (czlen == 0) return cz;
    if (czlen == 3) {
        // cz = [a,b,x] luon co cycle [a,b,c,a]
        cz.push_back(cz.front());
        return cz;
    }
    visit.assign(n+1, false); // cac dinh da tham
    VI v;
    v.assign(czlen, 0); // path

```


122 CHUYÊN TIN 11: CÁC CHUYÊN ĐỀ CHỌN LỌC VỚI NGÔN NGỮ C++

```

int x = cz[0], k = 0;
/* xuất phát từ cz[0]
   tìm đường đi x->x1->x2->...->xk
   với mọi xk tìm cạnh (xi,xk) i < k-1
   để tạo chu trình
*/
v[k] = x; // xuất phát từ đỉnh x
visit[x] = true; // đã thăm đỉnh x
while (true) {
    VI vv = Goal(v);
    if (vv.size() > 3) return vv; // tìm được chu trình
    if (k < 0) return Empty;
    int z = Next(cz, v[k]); // v[k] -> z ?
    if (z > 0) { // tìm được
        v[++k] = z;
        visit[z] = 1; // đã thăm đỉnh z
    } else --k; // lùi
} // while
}

VI Input() {
    ifstream f(FN);
    f >> n >> m;
    cout << n << " đỉnh, " << m << " cạnh " << endl;
    // Khởi tạo tập đỉnh dùng cho Find-Union
    for (int i = 0; i <= n; ++i) d.push_back(i);
    VI cz;
    // Khởi tạo ma trận kề a toàn 0
    for (int i = 0; i <= n; ++i) {
        VC b; b.assign(n+1, 0); a.push_back(b);
    }
    int xx = 0;
    int x, y; // cạnh (x,y)
    for(int i = 1; i <= m; ++i) { // cạnh (x,y)
        f >> x >> y;
        if (x == y) continue;
        a[x][y] = a[y][x] = 1;
        if (Union(x,y) == 0) { // gặp cycle

```

```

        xx = x; // ghi nhan dinh x cua canh khép kín cuối cùng
    }
} // for
if (xx == 0) return Empty; // không có cạnh khép kín tạo chu trình
// xác định TPLT chưa cycle
// đặt lại tro trực tiếp
for(int i = 1; i <= n; ++i) d[i] = Find(i);
int dx = d[xx];
for (int j = dx; j <= n; ++j) if (d[j] == dx) cz.push_back(j);
return cz;
}

void Run() {
    VI cz = Input();
    if (cz.size() == 0) { cout << 0 << endl; return;}
    cout << 1 << " : having a cycle";
    Print(cz, "\n Connected zone with a cycle: ");
    VI cle = Cycle(cz);
    Print(cle, "\n Found a Cycle:");
}

int main() {
    Run();
    cout << "\n T h e   E n d";
    return 0;
}

```

Output

```

10 dinh, 10 canh
1 : có cycle
Conneced zone with a cycle:  3 4 6 7 8
Found a Cycle: 3 4 8 6 3
T h e   E n d

```

Cycles in Spanning Forest

Khác với tiếp cận back tracking trong mục trước, trong mục này chúng ta tìm tại mỗi TPLT (zone) một cycle (nếu có) theo thuật toán *rút gọn tập đỉnh* như sau:

- ☞ Trước hết đọc dữ liệu cho đồ thị G gồm N đỉnh, M cạnh E từ input file. $O(M)$.
- ☞ Dùng kỹ thuật find-union FU xác định từng vùng liên thông zone gồm các đỉnh của G . Mỗi zone sẽ là một vùng liên thông của G . $O(MN)$.

124 CHUYÊN TIN 11: CÁC CHUYÊN ĐỀ CHỌN LỌC VỚI NGÔN NGỮ C++

- ✧ Với mỗi zone xây dựng một subgraph $g = (g_v, g_e)$, trong đó $g_v = \text{zone}$ là tập đỉnh, g_e là tập các cạnh (x, y) có hai đỉnh x và y trong zone. $O(M)$.
- ✧ Nếu g có cycle thì tìm một cycle trong g bằng thuật toán thu gọn đỉnh như sau:
 - Lần lượt duyệt các đỉnh v trong g_v . Nếu thử loại bỏ đỉnh v khỏi g_v mà vẫn thu được cycle từ các cạnh $e \in g_e$ không chứa đỉnh v thì loại bỏ hẳn v khỏi g_v . Ta gọi tính chất có cycle là *bất biến* của thuật toán thu gọn. Gọi (v_v, e_e) là subgraph thu được sau thủ tục thu gọn. $O(MN)$.
 - Từ (v_v, e_e) sắp xếp lại tập đỉnh v_v để thu được một cycle dạng $x \rightarrow y \rightarrow \dots \rightarrow z \rightarrow x$. $O(MN)$.

Program

```
// Cycle.CPP: cycle in connected zone cz
// Phuong an 2: duyet theo dinh
#include<bits/stdc++.h>
using namespace std;
typedef vector<int> VI;
int N, M; // number of vertices, edges
const char * FN = "GRAPH.INP";
class Edge {
public:
    int X, Y; // Edge (x,y)
    Edge(int x, int y) { // condition: x < y
        if(x < y) { X = x; Y = y; } else { X = y; Y = x; }
    }
    friend operator ==(Edge a, Edge b) { return a.X == b.X && a.Y == b.Y; }
    friend operator !=(Edge a, Edge b) { return !(a == b); }
    void Print(const char *msg="") { cout <<msg<<" ("<<X<<","<<Y<<")"; }
};
typedef vector<Edge> VE;
VE E; // edges
void Print(VI v, const char * msg = "") {
    cout << msg;
    for (int i = 0; i < v.size(); ++i) cout << " " << v[i];
}
// init and return poiter of vertices d
VI Init() {
    VI d(N+1);
```

```

    for(int i = 0; i <= N; ++i) d[i] = i;
    return d;
}
// head of set containing vertex x. O(n)
int Find(const VI &d, int x) {
    while (d[x] != x) x = d[x];
    return x;
}
// Union two subsets containing x and y. O(n)
int Union(VI & d, int x, int y) {
    x = Find(d, x); y = Find(d, y);
    if (x == y) return 0;
    if (x < y) d[y] = x; else d[x] = y;
    return 1;
}
// change d[i] = find(i)
// compute d[0] = number of zones
// return d[0]
// O(n^2)
int Update(VI &d) {
    int c = 0;
    for(int i = 1; i <= N; ++i) {
        if (d[i] == i) {
            ++c;
            for(int j = i+1; j <= N; ++j)
                d[j] = Find(d, j);
        }
    }
    d[0] = c;
    return c;
}
// Find-Union on E
VI FU() {
    VI d = Init();
    for(auto e:E) Union(d, e.X, e.Y);
    Update(d);
    return d;
}

```

```

void Input() {
    cout << " Input: " << endl;
    ifstream f(FN);
    if(!f) {cout << "\n unable open input file " << FN;
        exit(0);
    }
    f >> N >> M;
    cout << N << " vertices, " << M << " edges" << endl;
    E.clear();
    int x, y;
    for(int i = 1; i <= M; ++i) {
        f >> x >> y;
        if (x == y) continue;
        Edge e(x,y);    // x != y
        E.push_back(e);
    }
    f.close();
}

// Edge e in vertices z ?
bool IsIn(const VI &z, Edge e) {
    return count(z.begin(), z.end(), e.X)
        && count(z.begin(), z.end(), e.Y);
}

// subgraph g = (gv,ge) has a cycle
bool HasCycle(const VI &gv, const VE &ge) {
    VI d = Init();
    for(auto e:ge) if (Union(d, e.X, e.Y) == 0) return 1; // has cycle
    return 0; // no cycle
}

// thu bo dinh v khoi subgraph (gvmge)
// va thu duoc cycle
bool Try(const VI &gv, const VE &ge, int v) {
    VI d = Init();
    for(auto e:ge) {
        if(e.X == v || e.Y == v) continue;
        if (Union(d, e.X, e.Y) == 0) return 1;
    }
}

```

```

    return 0;
}
// x -> y -> ... -> z -> x
VI ToCycle(const VI & vv, const VE & ee) {
    // Tao chu trinh tu tap dinh gv
    VI cycle;
    if(vv.size() == 3) {
        for(auto v:vv) cycle.push_back(v); // copy
        cycle.push_back(vv[0]);
        return cycle;
    }
    int m = ee.size();
    VI mark(m); // danh dau cac canh trong ge
    mark.assign(m, 0);
    for(int i = 0; i < m; ++i) mark[i] = 1;
    Edge e = ee[0];
    int x = e.X, y;
    cycle.push_back(x);
    int z = e.Y;
    cycle.push_back(z);
    mark[0] = 0;
    for(int i = 1; i <= m; ++i) {
        for(int j = 1; j <= m; ++j) {
            if (mark[j] == 0) continue;
            Edge e = ee[j];
            if(e.X == z || e.Y == z) {
                z = (e.X == z) ? e.Y : e.X;
                cycle.push_back(z);
                mark[j] = 0; break;
            }
        }
        // e
    }
    // i
    return cycle;
}
// find a cycle in subgraph (gv,ge) by reduce gv
VI GetCycle(const VI & gv, const VE & ge) {
    if(gv.size() == 3) return ToCycle(gv, ge);

```

```

    VI mark(N+1); // danh dau cac dinh hien co
    mark.assign(N+1, 0);
    for(auto v:gv) mark[v] = 1;
    // reduce gv
    for(auto v:gv) if (Try(gv, ge, v)) mark[v] = 0; // erase v
    VI vv;
    for(auto v:gv) if (mark[v]) vv.push_back(v);
    VE ee;
    for(auto e:ge) if(IsIn(vv, e)) ee.push_back(e);
    return ToCycle(vv, ee);
}

// Find a cycle in edges ze
VI Cycle(const VI & gv) {
    // subgraph g = (gv, ge)
    VE ge;
    for(auto e:E) if(IsIn(gv, e)) ge.push_back(e);
    VI cycle;
    if (HasCycle(gv, ge)) cout << "\n has a cycle";
    else { cout << "\n no cycle"; return cycle; }
    // find a cycle for (gv, ge)
    return GetCycle(gv, ge);
}

void Run() {
    Input();
    for(auto x:E) x.Print("\n ");
    // Find-Union to separate to zones
    VI d = FU();
    cout << "\n number of zones: " << d[0];
    int c = 0; // counter
    for(int i = 1; i <= N; ++i) { // scan vertices
        if (d[i] == i) { // i is head
            VI zone; // vertices in zone i
            for(int j = i; j <= N; ++j)
                if (d[j] == i) zone.push_back(j);
            ++c;
            cout << "\n zone No " << c;
            VI cycle = Cycle(zone);

```

```

        cout << "\n Found:";
        for(auto x:cycle) cout << " " << x;
    } // zone
} // for
}
int main() {
    Run();
    cout << "\n T h e   E n d";
    return 0;
}

```

Output

```

Input:
10 vertices, 10 edges
(1, 9)
(2, 5)
(4, 7)
(6, 8)
(1, 5)
(3, 4)
(9, 10)
(2, 10)
(4, 8)
(3, 6)
number of zones: 2
zone No 1
has a cycle
Found: 1 9 10 2 5 1
zone No 2
has a cycle
Found: 6 8 4 3 6
T h e   E n d

```

LAN

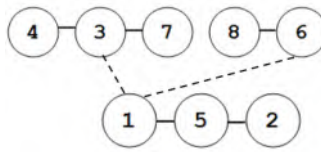
Một phòng học có n máy tính mang mã số từ 1 đến n , trong đó có m cặp máy tính được nối với nhau. Hãy cho biết cần nối thêm ít nhất bao nhiêu cặp máy tính để tạo thành một mạng cục bộ (Local Area Network - LAN) theo nghĩa hai máy tính bất kỳ có thể trực tiếp gửi thông điệp cho nhau hoặc thông qua một số máy tính trung gian.

130 CHUYÊN TIN 11: CÁC CHUYÊN ĐỀ CHỌN LỌC VỚI NGÔN NGỮ C++

Net.inp	Output
8 5	2
2 5	1 3
3 7	1 6
8 6	
5 1	
4 3	

Input: 8 máy tính, 5 cặp máy được nối với nhau.

Output: Cần nối thêm 2 cặp máy 1-3 và 1-6.



Algorithm

Quan sát ví dụ trong đề bài ta dễ dàng phát hiện ra thuật toán:

Bước 1. Gọi thuật toán Find–Union để cập nhật mảng tham chiếu d và tính số thành phần liên thông k

Bước 2. Quyết định:

- ✧ Nếu $k = 1$, tức là đồ thị là liên thông thì không cần nối thêm cặp máy tính nào.
- ✧ Nếu $k > 1$ thì nối thêm $k-1$ cặp máy tính $(1, i)$, trong đó i là "nhóm trưởng". Để ý rằng 1 là số hiệu nhỏ nhất nên 1 luôn luôn là nhóm trưởng, do đó ta chỉ cần nối máy 1 với các máy là nhóm trưởng còn lại.

Program

```

// LAN
#include<bits/stdc++.h>
using namespace std;
const char * FN = "GRAPH.INP";
vector<int> d;
int Find(int x) {
    while (d[x] != x) x = d[x];
    return x;
}
int Union(int x, int y) {
    x = Find(x); y = Find(y);
    if (x == y) return 0;
    if (x < y) d[y] = x; else d[x] = y;
    return 1;
}
void LAN() {
    ifstream f(FN);
    if(!f) {

```

```

    cout << "\n unable open input file " << FN;
    exit(0);
}
int n, m;
f >> n >> m;
d.assign(n+1, 0);
for (int i = 0; i <= n; ++i) d[i] = i;
int x, y;
int k = n; // so TPLT
for (int i = 1; i <= m; ++i) {
    f >> x >> y;
    if (x == y) continue;
    k -= Union(x, y);
}
f.close();
// chinh lai tro truc tiep
for (int i = 0; i <= n; ++i) d[i] = Find(i);
cout << "\n Can noi them " << k-1 << " cap may tinh";
for (int i = 2; i <= n; ++i)
    if (d[i] == i) cout << "\n 1 - " << i;
}
int main() {
    LAN();
    cout << "\n T h e   E n d";
    return 0;
}

```

Output

```

Can noi them 2 cap may tinh
1 - 3
1 - 6
T h e   E n d

```

Water Valves

ACM

Có N thùng nước được đánh số từ 1 đến N, giữa 2 thùng bất kỳ đều có một ống nối, trên ống có một van có thể đóng (khóa: không cho nước lưu thông trong ống đó) hoặc mở cho nước chảy trong ống. Ở trạng thái ban đầu tất cả các van đều đóng.

132 CHUYÊN TIN 11: CÁC CHUYÊN ĐỀ CHỌN LỌC VỚI NGÔN NGỮ C++

Bạn được cho một số yêu cầu thuộc 2 dạng với ý nghĩa như sau:

Dạng XY 1: bạn cần mở van trên ống nối giữa 2 thùng X và Y.

Dạng XY 2: bạn cần cho biết với trạng thái các van đang mở/đóng như hiện tại thì 2 thùng X và Y có thuộc cùng một nhóm bình thông nhau hay không, nghĩa là nước từ bình này có thể chảy đến được bình kia qua một số ống có van đang mở.

Input text file:

- ☞ Dòng đầu tiên ghi số thùng nước N và số yêu cầu P, $1 \leq N \leq 10000$, $1 \leq P \leq 50000$.
- ☞ Trong P dòng tiếp theo, mỗi dòng ghi ba số nguyên dương X, Y, Z với ý nghĩa có yêu cầu loại Z ($Z = 1$ hoặc $Z = 2$) với 2 thùng X và Y.

Output:

- ☞ Với mỗi yêu cầu dạng XY 2 ($Z = 2$) bạn cần ghi trên một dòng một trong 2 kết quả 0 hoặc 1 với ý nghĩa như sau:
 - 0: hai thùng X và Y thông nhau;
 - 1: hai thùng X và Y không thông nhau.

Ví dụ

VALVES.INP	Output	
7 9		7 thùng, 9 yêu cầu
1 2 2	0	Thùng 1 và 2 không liên thông
1 2 1		
3 7 2	0	Thùng 3 và 7 không liên thông
2 3 1		
1 3 2	1	Thùng 1 và 3 liên thông
2 4 2	0	Thùng 2 và 4 không liên thông
1 4 1		
3 4 2	1	Thùng 3 và 4 liên thông
1 7 2	0	Thùng 1 và 7 không liên thông

Algorithm

Bài này khá đơn giản. Hàm Van dưới đây đọc dần từng dòng lệnh x y z và xét:

- ☞ Nếu $z = 1$: thực hiện Union(x,y) để gộp TPLT chứa thùng x với TPLT chứa thùng y cho kết quả là TPLT chứa đồng thời hai thùng x và y.
- ☞ Nếu $z = 2$: Xét
 - Nếu $\text{Find}(x) = \text{Find}(y)$ tức là hai thùng x và y thuộc cùng một TPLT nên x và y thông nhau;
 - Nếu $\text{Find}(x) \neq \text{Find}(y)$ tức là hai thùng x và y thuộc hai TPLT khác nhau nên x và y không thông nhau.

Chú ý: Khi cài đặt, chúng ta thường quên gọi thủ tục khởi trị mảng tham chiếu d!!!

Program

```

// water Valves
#include<bits/stdc++.h>
using namespace std;
const char * FN = "VALVES.INP";
vector<int> d;
int Find(int x) {
    while (d[x] != x) x = d[x];
    return x;
}
int Union(int x, int y) {
    x = Find(x); y = Find(y);
    if (x == y) return 0;
    if (x < y) d[y] = x; else d[x] = y;
    return 1;
}
void Valves() {
    ifstream f(FN);
    if(!f) {
        cout << "\n unable open input file " << FN;
        exit(0);
    }
    int n, m;
    f >> n >> m;
    d.clear();
    for (int i = 0; i <= n; ++i) d.push_back(i);
    int x, y, z;
    int k = n; // so TPLT
    for (int i = 1; i <= m; ++i) {
        f >> x >> y >> z;
        if (x == y) continue;
        if (z == 1) Union(x,y);
        else { // z == 2: ask
            if (Find(x) == Find(y)) cout << "\n 1";
            else cout << "\n 0";
        }
    }
}

```

134 CHUYÊN TIN 11: CÁC CHUYÊN ĐỀ CHỌN LỌC VỚI NGÔN NGỮ C++

```
f.close();
}
int main() {
    Valves();
    cout << "\n T h e    E n d";
    return 0;
}
```

Bridges

Một cạnh (x,y) trong đồ thị vô hướng G được gọi là cầu hoặc cạnh trọng yếu nếu xóa cạnh đó đi (giữ lại các đỉnh x và y ở hai đầu), sẽ làm tăng số thành phần liên thông của đồ thị. Hãy liệt kê các cầu của đồ thị n đỉnh và m cạnh.

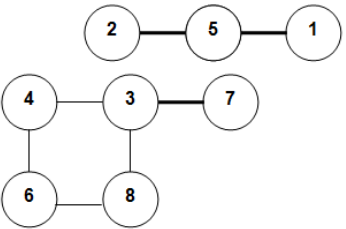
Ví dụ

GRAPH. INP	Output
8 7	3
3 7	1 5
1 5	2 5
3 4	3 7
2 5	
3 8	
4 6	
6 8	

Input: Đồ thị có 8 đỉnh, 7 cạnh.

Output: Đồ thị có 3 cầu (1,5), (2,5) và (3,7)

(tô đậm)



Algorithm

Nhận xét 1: Mọi cầu phải là cạnh khung, nghĩa là phải thuộc một cây hoặc rừng khung.

Thật vậy, nếu cạnh (x,y) không là cạnh khung thì khi xét cạnh đó trong thủ tục tìm cây khung ta phải có $\text{Union}(x,y) = 0$, thế tức là đồ thị có chu trình chứa cạnh (x,y) này. Khi bỏ cạnh (x,y) trong chu trình ta thấy các đỉnh nằm trên chu trình này vẫn liên thông với nhau, do đó số thành phần liên thông là không đổi. Điều này cho thấy (x,y) không thể là cầu.

Chú ý rằng mệnh đề đảo của mệnh đề trên không đúng. Cạnh (3,4) của hình trong ví dụ có thể nằm trong rừng khung nhưng (3,4) không phải là cầu.

Ta gọi bậc của đỉnh x trong đồ thị vô hướng là số cạnh đi ra (hoặc đi đến) đỉnh đó.

Nhận xét 2: Nếu một cạnh có một đầu bậc 1 thì cạnh đó là cầu.

Thật vậy, giả sử cạnh (x,y) có bậc của đỉnh x là 1 thì khi xóa cạnh đó ta biến đỉnh x thành đỉnh cô lập tức là x được tách thành một TPLT, do đó số TPLT được tăng lên.

Tuy nhiên, dễ nhận thấy rằng cạnh có đầu bậc 1 thì nằm trong rừng khung.

Hai nhận xét trên cho phép ta chỉ kiểm tra những cạnh khung để phát hiện cầu.

✂ *Bước 1.* Đọc dữ liệu, dùng kỹ thuật Find-Union ghi nhận các giá trị sau đây:

- số mảnh liên thông.
- mảng chứa các cạnh khung.
- Tính bậc của các đỉnh.

Độ phức tạp: $O(nm)$

✂ *Bước 2.* Duyệt các cạnh khung (x,y) .

- Nếu bậc của x hoặc y là 1 thì đưa vào cầu.
- Nếu thử bỏ cạnh (x, y) mà số thành phần liên thông tăng lên thì (x,y) là cầu, Chú ý rằng việc tính bậc của đỉnh dù sao cũng đơn giản hơn việc thử bỏ cạnh khung.

Độ phức tạp: $O(nm^2)$

Program

```
// Bridges
#include<bits/stdc++.h>
using namespace std;
const char * FN = "GRAPH.INP";
class Edge {
public:
    int X, Y; // cạnh (x,y)
    Edge(int x, int y) {
        if(x < y) { X = x; Y = y; }
        else { X = y; Y = x; }
    }
    bool operator ==(const Edge & c) { return X == c.X && Y == c.Y; }
    bool operator !=(const Edge & c) { return !(*this == c); }
    void Print(const char * msg = "") {
        cout << msg << " (" << X << ", " << Y << ")";
    }
};
typedef vector<Edge> VE;
VE E;
typedef vector<int> VI;
VI d, deg; // tro dinh, bac
```

```

// label of Edges
// 1: canh khung; 2: cau
VI label;
int N, M; // so dinh, so canh
int K; // so TPLT (so zone)
int Find(int x) { while (d[x] != x) x = d[x]; return x; }
int Union(int x, int y) {
    x = Find(x); y = Find(y);
    if (x == y) return 0;
    if (x < y) d[y] = x; else d[x] = y;
    return 1;
}
void Initd() { for(int i = 0; i <= N; ++i) d[i] = i; }
int Updated() {
    for (int i = 1; i <= N; ++i) d[i] = Find(i);
    d[0] = 0;
    for (int i = 1; i <= N; ++i) if(d[i] == i) ++d[0];
    return d[0];
}
inline int Union(Edge c) { return Union(c.X, c.Y); }
void Input() {
    ifstream f(FN);
    if(!f) {
        cout << "\n unable open input file " << FN;
        exit(0);
    }
    f >> N >> M;
    E.clear();
    int x, y;
    for(int i = 1; i <= M; ++i) {
        f >> x >> y;
        // cout << "\n " << x << ", " << y;
        if(x == y) continue;
        E.push_back(Edge(x, y));
    }
    f.close();
}

```

```

// thu bo canh thu i khoi E
// co lam tang TPLT ?
bool Try(int i) {
    // hien co K TPLT
    // E[i].Print("\n xet canh " );
    Initd();
    for(int j = 0; j < M; ++j) if(j != i) Union(E[j].X, E[j].Y);
    Updated(); return K < d[0];
}

// return vector of Edge
VE Bridges() {
    Input();
    cout << "\n " << N << " vertices, " << M << " edges";
    d.assign(N+1, 0); // vertices pointer
    deg.assign(N+1, 0); // vertices pointer
    label.assign(M, 0); // label of edges
    Initd();
    cout << "\n Edges of G: ";
    for(int i = 0; i < M; ++i) {
        int x = E[i].X, y = E[i].Y;
        E[i].Print("\n ");
        ++deg[x]; ++deg[y];
        label[i] = Union(x, y);
        cout << ": " << label[i];
    }
    Updated();
    cout << "\n Vertices of Degree 1: ";
    for(int i = 0; i <= N; ++i) if (deg[i] == 1) cout << " " << i;
    K = d[0];
    cout << "\n zones: " << K;
    // Duyệt lại các cạnh để tìm cầu
    for(int i = 0; i < M; ++i) {
        // nếu là cạnh (0) thì bỏ qua
        if(label[i] == 0) continue; // i không là cạnh khung
        int x = E[i].X, y = E[i].Y;
        // nếu bậc đỉnh == 1 thì đưa vào cầu
        if(deg[x] == 1 || deg[y] == 1) {

```


138 CHUYÊN TIN 11: CÁC CHUYÊN ĐỀ CHỌN LỌC VỚI NGÔN NGỮ C++

```
        label[i] = 2; // cau
        continue;
    }
    // label = 1 (khung)
    // thu bo canh i co lam tang TPLT ?
    if (Try(i)) label[i] = 2;
}
VE b; // bridges
for(int i = 0; i < M; ++i) if(label[i] == 2) b.push_back(E[i]);
return b;
}
int main() {
    VE b = Bridges();
    cout << "\n Final Bridges: ";
    for(auto e:b) e.Print("\n ");
    cout << "\n T h e   E n d";
    return 0;
}
```

Output

```
8 vertices, 7 edges
Edges of G:
(3, 7): 1
(1, 5): 1
(3, 4): 1
(2, 5): 1
(3, 8): 1
(4, 6): 1
(6, 8): 0
Vertices of Degree 1:  1 2 7
zones: 2
Final Bridges:
(3, 7)
(1, 5)
(2, 5)
T h e   E n d
```

Test 2

Test này có 6 câu

GRAPH.INP	Output
19 18	6
9 1	(4, 7)
2 5	(11, 12)
4 7	(12, 13)
8 6	(13, 14)
5 1	(16, 18)
4 3	(16, 19)
9 10	
10 2	
4 8	
3 6	
11 12	
12 13	
13 14	
18 15	
18 16	
18 17	
16 19	
17 15	

Bình luận

1. Đề ý rằng nếu một cạnh thuộc chu trình thì cạnh đó không thể là cầu. Do tính chất khép kín của chu trình, khi thiếu một cạnh trong chu trình thì phần còn lại vẫn liên thông. Do đó, ta chỉ cần khảo sát các cạnh khung ngoài chu trình. Tuy nhiên thuật toán tìm chu trình cũng không đơn giản nên ta không theo tiếp cận này.

2. Hàm Try(i) duyệt lại toàn bộ M-1 cạnh trừ cạnh thứ i để thực hiện Find-Union xem có làm tăng số TPLT của đồ thị G. Ta có thể thực hiện Try trên mỗi mảnh liên thông (đồ thị con) thì số lần duyệt cạnh có thể giảm.

Ta cải tiến thuật toán như sau:

- ☞ Trước hết đọc dữ liệu cho đồ thị G N đỉnh, M cạnh E từ input file. $O(M)$.
- ☞ Dùng kỹ thuật Find-Union (FU) xác định từng vùng liên thông zone gồm các đỉnh của G. Mỗi zone sẽ là một vùng liên thông của G. $O(MN)$.
- ☞ Với mỗi zone xây dựng một subgraph $g = (g_v, g_e)$, trong đó $g_v = \text{zone}$ là tập đỉnh, g_e là tập các cạnh (x,y) có hai đỉnh x và y trong zone. $O(M)$.
 - Với mỗi subgraph $g = (g_v, g_e)$ tìm và đánh dấu các cạnh trong g_e là cầu.

Chương trình sau chạy với test 2.

Program

```

// Bridges
#include<bits/stdc++.h>
using namespace std;
const char * FN = "GRAPH.INP";
class Edge { . . . };
typedef vector<Edge> VE;
VE E;
typedef vector<int> VI;
VI deg; // vertex degree
// label of Edges
// 1: canh khung; 2: cau
VI label;
int N, M; // so dinh, so canh
int K; // so TPLT (so zone)
int Find(const VI &d, int x) { while (d[x] != x) x = d[x]; return x; }
int Union(VI & d, int x, int y) {
    x = Find(d, x); y = Find(d, y);
    if (x == y) return 0;
    if (x < y) d[y] = x; else d[x] = y;
    return 1;
}
inline int Union(VI & d, Edge e) { return Union(d, e.X, e.Y); }
// tao moi va khoi tri vertex pointers d
VI Init() {
    VI d(N+1);
    for(int i = 0; i <= N; ++i) d[i] = i;
    return d;
}
// reset tri cho vertex pointers d
void Reset(VI &d) { for(int i = 0; i <= N; ++i) d[i] = i; }
// cap nhat lai d
int Update(VI &d) {
    for (int i = 1; i <= N; ++i) d[i] = Find(d, i); // tro truc tiep
    d[0] = 0;
    for (int i = 1; i <= N; ++i) if(d[i] == i) ++d[0];
    return d[0]; // so TPLT
}

```

```

void Input() {
    ifstream f(FN);
    if(!f) {
        cout << "\n unable open input file " << FN;
        exit(0);
    }
    f >> N >> M; // so dinh, so canh
    E.clear();
    int x, y;
    for(int i = 1; i <= M; ++i) {
        f >> x >> y;
        if(x == y) continue;
        E.push_back(Edge(x, y));
    }
    f.close();
}

// thu bo canh thu i khoi subgraph g
// co lam tang TPLT ?
bool Try(const VI & gv, const VI & ge, int i) {
    // so vung lien thong hien co k
    int k = N-gv.size() + 1;
    // tong so dinh N
    // cac dinh (doc lap) khong nam trong subgraph = N-gv.size()
    // gv la mien LT
    VI d(N+1); Reset(d);
    for(auto j:ge) // xet tung canh j trong ge
        if (j != i) Union(d, E[j]);
    return k < Update(d);
}

// Edge e co 2 dinh trong v
bool IsIn(const VI & v, Edge e) {
    return count(v.begin(), v.end(), e.X)
        && count(v.begin(), v.end(), e.Y);
}

// label cac Bridges in subgraph g(gv, ge)
void Bridges(const VI & gv, const VI & ge) {
    cout << "\n subgraph (gv, ge): ";

```

```

        cout << "\n gv: ";
        for(auto x:gv) cout << " " << x;
        cout << "\n ge: ";
        for(auto i:ge) E[i].Print(" ");
// duyệt các cạnh trong ge
for(auto i:ge) {
    if(label[i] == 0) continue; // cạnh ngoài khung
    if(deg[E[i].X] == 1 || deg[E[i].Y] == 1) {
        label[i] = 2; // bridge
        continue;
    }
    // label[i] = 1: cạnh khung cần kiểm tra
    if(Try(gv, ge, i)) label[i] = 2;
}
}
// return vector of Edge
VE Bridges() {
    Input();
    cout << "\n " << N << " vertices, " << M << " edges";
    VI d = Init(); // vertex pointers
    deg.assign(N+1, 0); // vertex degree
    label.assign(M, 0); // label of edges
    cout << "\n Edges of G: ";
    for(int i = 0; i < M; ++i) {
        int x = E[i].X, y = E[i].Y;
        E[i].Print("\n ");
        ++deg[x]; ++deg[y];
        label[i] = Union(d, E[i]); // 0: cạnh thường, 1: cạnh khung
        cout << ": " << label[i];
    }
    K = Update(d);
    cout << "\n So TPLT: " << K;
    // Phân hoạch G thành các subgraph
    // g(gv, ge)
    for(int i = 0; i <= N; ++i) // duyệt các đỉnh
        if (d[i] == i) { // i is head
            // tạo subgraph cho zone i

```

```

    VI gv, ge; // cac dinh, cac index canh
    for(int j = i; j <= N; ++j) if (d[j] == i) gv.push_back(j);
    for(int i = 0; i < M; ++i) // duyet cac canh
        if(IsIn(gv, E[i])) ge.push_back(i);
    Bridges(gv, ge); // Bridges in subgraph g(gv, ge)
}
// tong ket
VE b; // biridges
for(int i = 0; i < M; ++i)
    if (label[i] == 2) b.push_back(E[i]);
return b;
}
int main() {
    VE b = Bridges();
    cout << "\n Final Bridges: " << b.size() << ": ";
    for(auto x:b) x.Print("\n ");
    cout << "\n T h e   E n d";
    return 0;
}

```

Output

19 vertices, 18 edges

Edges of G:

```

(1, 9): 1
(2, 5): 1
(4, 7): 1
(6, 8): 1*
(1, 5): 1
(3, 4): 1
(9, 10): 1
(2, 10): 0
(4, 8): 1
(3, 6): 0
(11, 12): 1
(12, 13): 1
(13, 14): 1
(15, 18): 1

```

144 CHUYÊN TIN 11: CÁC CHUYÊN ĐỀ CHỌN LỌC VỚI NGÔN NGỮ C++

```
(16, 18): 1
(17, 18): 1
(16, 19): 1
(15, 17): 0
subgraph (gv, ge):
gv: 1 2 5 9 10
ge: (1, 9) (2, 5) (1, 5) (9, 10) (2, 10) ?
subgraph (gv, ge):
gv: 3 4 6 7 8
ge: (4, 7) (6, 8) (3, 4) (4, 8) (3, 6) ?
subgraph (gv, ge):
gv: 11 12 13 14
ge: (11, 12) (12, 13) (13, 14) ?
subgraph (gv, ge):
gv: 15 16 17 18 19
ge: (15, 18) (16, 18) (17, 18) (16, 19) (15, 17) ?
Final Bridges: 6:
(4, 7)
(11, 12)
(12, 13)
(13, 14)
(16, 18)
(16, 19)
T h e   E n d
```

Simple version

Một phương án đơn giản là như sau:

- ☞ Input: Đọc dữ liệu, xác định số đỉnh N , số cạnh M của đồ thị $G = (V, E)$. Với mỗi cạnh (x, y) tính bậc của các đỉnh $\deg(x)$, $\deg(y)$, thực hiện $\text{Union}(x, y)$ để đánh dấu các cạnh khung là các cạnh có $\text{Union}(x, y) = 1$. Hàm Input cho ra số TPLT k .
- ☞ Bridges: Duyệt lại các cạnh khung (x, y) . Nếu $\deg(x)$ hoặc $\deg(y) = 1$ thì (x, y) là cầu hoặc thử bỏ cạnh này mà số TPLT tăng lên thì (x, y) là cầu.

Program

```
/*-----
Bridges: simple
-----*/
```

```

#include<bits/stdc++.h>
using namespace std;
const char * FN = "GRAPH.INP";
typedef vector<int > VI;
int N; // number of vertices
int M; // number of edges
VI d; // tro dinh
VI mark; // canh 2, khung 1, tao chu trinh 0
VI deg; // vertex degree
class Edge { . . . };
typedef vector<Edge> VE;
VE E; // edges
// init pointer to vertices d
void Init() {for(int i = 0; i <= N; ++i) d[i] = i; }
// head of set containing vertex x. O(n)
int Find(int x) { while (d[x] != x) x = d[x]; return x; }
// Union two subsets containing x and y. O(n)
int Union(int x, int y) {
    x = Find(x); y = Find(y);
    if (x == y) return 0;
    if (x < y) d[y] = x; else d[x] = y;
    return 1;
}
int Union(Edge e) {
    return Union(e.X, e.Y);
}
// change d[i] = find(i)
// compute d[0] = number of zones
// return d[0]
// O(n^2)
int Update() {
    for(int i = 1; i <= N; ++i) d[i] = Find(i);
    for(int i = 1; i <= N; ++i) if (d[i] == i) ++d[0];
    return d[0];
}
// FU on G excluding edge i-th
// return the number of zones

```



```

int FU(int i) {
    Init();
    for(int j = 0; j < M; ++j) if(j != i) Union(E[j]);
    Update();
    return d[0];
}

// read data and compute vertex degree
// union, mark spanning edge
// return number of connected zone
int Input() {
    cout << " Input: " << endl;
    ifstream f(FN);
    if(!f) {
        cout << "\n unable open input file " << FN;
        exit(0);
    }
    f >> N >> M;
    cout << N << " vertices, " << M << " edges" << endl;
    deg.assign(N+1, 0);
    mark.assign(M, 2);
    d.assign(N+1,0);
    Init(); // set d = 0 1 2 ... N
    // read Edges
    E.clear();
    int x, y;
    for(int i = 0; i < M; ++i) {
        f >> x >> y;
        Edge e(x,y);
        E.push_back(e);
        e.Print("\n ");
        ++deg[x]; ++deg[y];
        mark[i] = Union(x,y);
    }
    f.close();
    Update();
    return d[0];
}

```

```

void Bridges() {
    cout << "\n Brodges: ";
    int k = Input();
    cout << "\n Result: ";
    for(int i = 0; i < M; ++i) {
        if(mark[i] == 1) {
            if(deg[E[i].X] == 1 || deg[E[i].Y] == 1) {
                E[i].Print("\n by deg ");
            }
            // i is in spanning forest
            // FU excluding i
            else if(FU(i) > k) E[i].Print("\n ");
        }
    }
}

int main() {
    Bridges();
    cout << "\n T h e   E n d";
    return 0;
}

```

Output

```

Brodges:  Input:
19 vertices, 18 edges
(1, 9)
(2, 5)
(4, 7)
(6, 8)
(1, 5)
(3, 4)
(9, 10)
(2, 10)
(4, 8)
(3, 6)
(11, 12)
(12, 13)

```

```
(13, 14)
(15, 18)
(16, 18)
(17, 18)
(16, 19)
(15, 17)
Result:
by deg (4, 7)
by deg (11, 12)
(12, 13)
by deg (13, 14)
(16, 18)
by deg (16, 19)
T h e   E n d
```

Intersection Of Spanning Trees

Ta quy ước gọi khung là cây hoặc rừng khung của một đồ thị. Đồ thị có thể có một hoặc nhiều khung. Hãy liệt kê giao của mọi khung trong đồ thị vô hướng n đỉnh, m cạnh, tức là liệt kê các cạnh có mặt trong mọi khung của đồ thị.

Gợi ý: *Giao của khung = Tập các cầu.*

Joint Vertex

Đỉnh khớp của một đồ thị vô hướng là đỉnh thỏa tính chất sau: nếu ta xóa đỉnh đó và các cạnh liên thuộc với đỉnh đó sẽ làm tăng số thành phần liên thông của đồ thị. Hãy liệt kê các đỉnh khớp trong đồ thị vô hướng n đỉnh, m cạnh.

Chú ý: Bạn cần phân biệt các thuật ngữ sau:

1. Xóa cạnh x - y được hiểu là loại bỏ đường nối giữa hai đỉnh x và y nhưng giữ lại hai đỉnh đó. Về mặt trực quan ta hiểu như sau:

Trước khi xóa cạnh x - y : $(x) \text{---} (y)$

Sau khi xóa cạnh x - y : $(x) \quad (y)$

2. Xóa đỉnh x được hiểu là loại bỏ đỉnh x và mọi cạnh liên thuộc đỉnh đó. Về mặt trực quan ta hiểu như sau:

Trước khi xóa đỉnh x : $(z) \text{---} (x) \text{---} (y)$

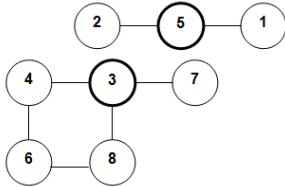
Sau khi xóa đỉnh x : $(z) \quad (y)$

Ví dụ

Input	Output
8 7	2
1 5	3
2 5	5
3 7	
3 4	
3 8	
4 6	
6 8	

Input: Đồ thị có 8 đỉnh, 7 cạnh.

Output: Đồ thị có 2 đỉnh khớp: 3 và 5. (tô đậm)



Algorithm

Nhận xét: Ta gọi bậc của đỉnh v trong đồ thị vô hướng là số cạnh đi ra (hoặc đi đến) đỉnh đó. Dễ thấy là nếu đỉnh v có bậc < 2 thì v không thể là đỉnh khớp.

- ☞ Đọc dữ liệu cho đồ thị G N đỉnh, M cạnh E từ input file. $O(M)$.
- ☞ Dùng kỹ thuật Find-Union (FU) xác định từng vùng liên thông zone gồm các đỉnh của G . Mỗi zone sẽ là một vùng liên thông của G . $O(MN)$.
- ☞ Với mỗi zone xây dựng một subgraph $g = (g_v, g_e)$, trong đó $g_v = \text{zone}$ là tập đỉnh, g_e là tập các cạnh (x,y) có hai đỉnh x và y trong zone. $O(M)$.
 - Với mỗi subgraph $g = (g_v, g_e)$ tìm và đánh dấu các đỉnh joint trong g_v .

Độ phức tạp: $O(n^2m^2)$

Program

```
// JV Joint Vertices
#include<bits/stdc++.h>
using namespace std;
const char * FN = "GRAPH.INP";
class Edge { . . . };
typedef vector<Edge> VE;
VE E;
typedef vector<int> VI;
VI deg; // degree of vertex
// label of vertices, 0: not Joint, 1: joint
VI label;
int N, M; // so dinh, so canh
int K; // so TPLT (so zone)
```

150 CHUYÊN TIN 11: CÁC CHUYÊN ĐỀ CHỌN LỌC VỚI NGÔN NGỮ C++

```
int Find(const VI &d, int x) { while (d[x] != x) x = d[x]; return x; }
int Union(VI & d, int x, int y) {
    x = Find(d, x); y = Find(d, y);
    if (x == y) return 0;
    if (x < y) d[y] = x; else d[x] = y;
    return 1;
}
inline int Union(VI & d, Edge e) { return Union(d, e.X, e.Y); }
// tao moi va khoi tri vertex pointers d
VI Init() {
    VI d(N+1);
    for(int i = 0; i <= N; ++i) d[i] = i;
    return d;
}
// reset tri cho vertex pointers d
void Reset(VI &d) { for(int i = 0; i <= N; ++i) d[i] = i; }
// cap nhat lai d
int Update(VI &d) {
    for (int i = 1; i <= N; ++i) d[i] = Find(d, i); // tro truc tirp
    d[0] = 0;
    for (int i = 1; i <= N; ++i) if(d[i] == i) ++d[0];
    return d[0]; // so TPLT
}
void Input() {
    ifstream f(FN);
    if(!f) { cout << "\n unable open input file " << FN;
        exit(0);
    }
    f >> N >> M; // so dinh, so canh
    E.clear();
    int x, y;
    for(int i = 1; i <= M; ++i) {
        f >> x >> y; if(x == y) continue;
        E.push_back(Edge(x, y));
    }
    f.close();
}
```

```

// Edge e co 2 dinh trong v
bool IsIn(const VI & v, Edge e) {
    return count(v.begin(), v.end(), e.X)
        && count(v.begin(), v.end(), e.Y);
}
// thu bo vertex i khoi subgraph g
// co lam tang TPLT ?
bool Try(const VI & gv, const VE & ge, int i) {
    // so vung lien thong hien co k
    int k = N-gv.size() + 1;
    // tong so dinh N
    // cac dinh (doc lap) khong nam trong subgraph = N-gv.size()
    // gv la mien LT
    VI d(N+1); Reset(d);
    for(auto e:ge) { // xet tung canh j trong ge
        if(e.X != i && e.Y != i) Union(d, e);
    }
    return k < Update(d)-1; // ban than dinh i cung bi xoa
}
// label cac joint verticed in subgraph g(gv, ge)
void JV(const VI & gv, const VE & ge) {
    cout << "\n subgraph (gv, ge): ";
    cout << "\n gv: ";
    for(auto x:gv) cout << " " << x;
    cout << "\n ge: ";
    for(auto e:ge) e.Print(" ");
    for(int i:gv) {
        if(deg[i] < 2) continue;
        // deg >= 2
        if(Try(gv,ge,i)) label[i] = 1;
    }
}
// return vector of Joint Vertices
VI JV() {
    Input();
    cout << "\n " << N << " vertices, " << M << " edges";
    VI d = Init(); // vertex pointers

```

```

deg.assign(N+1, 0); // vertex deggree
label.assign(N+1, 0); // label of vertices
cout << "\n Edges of G: ";
for(auto e:E) {
    e.Print("\n ");
    ++deg[e.X]; ++deg[e.Y];
    Union(d,e);
}
K = Update(d);
cout << "\n So TPLT: " << K;
// Phan hoach G thanh cac subgraph
// g(gv,ge)
for(int i = 0; i <= N; ++i) // duyet cac dinh
    if (d[i] == i) { // i is head
        // tao subgraph cho zone i
        VI gv;
        VE ge; // cac dinh, cac canh
        for(int j = i; j <= N; ++j)
            if (d[j] == i) gv.push_back(j);
        for(auto e:E) // duyet cac canh
            if(IsIn(gv, e)) ge.push_back(e);
        JV(gv, ge); // Mark joint vertices in subgraph g(gv, ge)
    }
// tong ket
VI b; // joint Vertices
for(int i = 1; i <= N; ++i) if(label[i]) b.push_back(i);
return b;
}

int main() {
    VI b = JV();
    cout << "\n All Joint Vertices in G " << b.size() << ": ";
    for(auto j:b) cout << " " << j;
    cout << "\n T h e   E n d";
    return 0;
}

```

Output theo dữ liệu đề bài

8 vertices, 7 edges

Edges of G:

(1, 5)

(2, 5)

(3, 7)

(3, 4)

(3, 8)

(4, 6)

(6, 8)

So TPLT: 2

subgraph (gv, ge):

gv: 1 2 5

ge: (1, 5) (2, 5)

subgraph (gv, ge):

gv: 3 4 6 7 8

ge: (3, 7) (3, 4) (3, 8) (4, 6) (6, 8)

All Joint Vertices in G 2: 3 5

T h e E n d

Output theo Test 2

19 vertices, 18 edges

Edges of G:

(1, 9)

. . .

So TPLT: 4

subgraph (gv, ge):

gv: 1 2 5 9 10

ge: (1, 9) (2, 5) (1, 5) (9, 10) (2, 10)

subgraph (gv, ge):

gv: 3 4 6 7 8

ge: (4, 7) (6, 8) (3, 4) (4, 8) (3, 6)

subgraph (gv, ge):

gv: 11 12 13 14

ge: (11, 12) (12, 13) (13, 14)

subgraph (gv, ge):

gv: 15 16 17 18 19

ge: (15, 18) (16, 18) (17, 18) (16, 19) (15, 17)

All Joint Vertices in G 5: 4 12 13 16 18

T h e E n d

Classroom LAN

Một phòng học có n máy tính mang mã số từ 1 đến n , trong đó có m cặp máy tính được nối với nhau. Hãy cho biết cần gỡ đi nhiều nhất bao nhiêu đoạn cáp nối các cặp máy tính và cần nối thêm ít nhất là bao nhiêu cặp máy để tạo thành một mạng theo nghĩa hai máy tính bất kỳ có thể trực tiếp gửi thông điệp cho nhau hoặc thông qua một số máy tính trung gian.

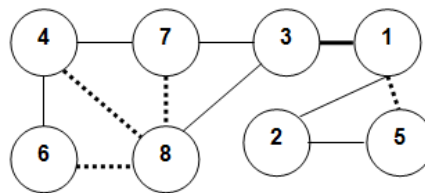
GRAPH. INP	Output
8 10	-4
2 5	1 5
1 2	4 8
1 5	6 8
3 7	7 8
3 8	+1
4 6	1 3
4 7	
4 8	
6 8	
7 8	

Input: 8 máy tính, trong đó có 10 cặp máy được nối với nhau.

Output: Cần gỡ 4 đoạn cáp 1-5, 4-8, 6-8 và 7-8 (cạnh ...);

Nối thêm 1 cặp máy 1-3 (cạnh tô đậm).

Chú ý: Lúc đầu Input không có cạnh 1-3.



Algorithm

Quan sát hình vẽ trong ví dụ ta dễ dàng phát hiện ra thuật toán:

- ✧ Bước 1. Đọc từng cạnh (x,y) , thực hiện Find-Union trên cạnh này. Ghi nhận cạnh (x,y) vào vector dc nếu $\text{Union}(x,y) = 0$ (tức là cạnh (x,y) nằm ngoài khung)
- ✧ Cập nhật mảng tham chiếu d, tính số thành phần liên thông K.
- ✧ Bước 2. Quyết định:
 - Các cạnh cần gỡ bớt dc
 - Số cạnh cần nối thêm $= K - 1$

Program

```
// classroom LAN
#include<bits/stdc++.h>
using namespace std;
const char * FN = "GRAPH.INP";
class Edge { . . . };*
typedef vector<Edge> VE;
typedef vector<int> VI;
```

```

int N, M; // so dinh, so canh
int K; // so TPLT (so zone)
int Find(const VI &d, int x) {
    while (d[x] != x) x = d[x];
    return x;
}
int Union(VI & d, int x, int y) {
    x = Find(d, x); y = Find(d, y);
    if (x == y) return 0;
    if (x < y) d[y] = x; else d[x] = y;
    return 1;
}
inline int Union(VI & d, Edge e) { return Union(d, e.X, e.Y); }
// tao moi va khoi tri vertex pointers d
VI Init() {
    VI d(N+1);
    for(int i = 0; i <= N; ++i) d[i] = i;
    return d;
}
// cap nhat lai d
int Update(VI &d) {
    for (int i = 1; i <= N; ++i) d[i] = Find(d, i); // tro truc tiep
    d[0] = 0;
    for (int i = 1; i <= N; ++i) if(d[i] == i) ++d[0];
    return d[0]; // so TPLT
}
void LAN() {
    ifstream f(FN);
    if(!f) {
        cout << "\n unable open input file " << FN;
        exit(0);
    }
    f >> N >> M; // so dinh, so canh
    cout << "\n " << N << " vertices, " << M << " edges";
    VI d = Init(); // vertex pointers
    VE dc; // disconnect edges
    cout << "\n Edges of G: ";

```

```

    int x, y;
    for(int i = 1; i <= M; ++i) {
        f >> x >> y; if(x == y) continue;
        Edge e(x, y); e.Print("\n ");
        if (!Union(d,e)) dc.push_back(e);
    }
    f.close();
    K = Update(d);
    cout << "\n Number of zones: " << K;
    cout << "\n Disconnect: -" << dc.size() << endl;
    for(auto e:dc) e.Print("\n ");
    VI head; // cac truong nhom
    for(int i = 1; i <= N; ++i) if(d[i] == i) head.push_back(i);
    cout << "\n Connect: +" << K-1;
    for(int i = 1; i < head.size(); ++i) cout << "\n 1 " << head[i];
}

int main() {
    LAN();
    cout << "\n T h e   E n d";
    return 0;
}

```

Output

```

8 vertices, 10 edges
Edges of G:
(2, 5)
. . .
So TPLT: 2
Disconnect: -4
(1, 5)
(4, 8)
(6, 8)
(7, 8)
Connect: +1
1 3
T h e   E n d

```

IT Teams

Có n bạn trẻ đến dự lễ hội Tin học. Khi vào cửa, mỗi bạn được một robot cấp một mã gồm 6 chữ cái in hoa 'A'...'Z'. Tại hội trường, Ban Tổ chức yêu cầu các bạn đứng theo từng đội để tham gia đấu trí. Hai người có mã giống nhau dù chỉ một chữ cái sẽ vào cùng một đội. Ví dụ, bạn trẻ 'ABCDAB' và bạn trẻ 'XXDEZEV' sẽ cùng vào một đội. Nếu A và B cùng đội, B và C cùng đội thì đương nhiên là ba bạn đó sẽ cùng đội. Hãy hiển thị trên màn hình số đội và liệt kê số hiệu của từng người trong đội tính từ 1 đến n .

Input text file "Team.inp": Dòng đầu tiên số n ; $\max(n) = 2000$;

Dòng thứ i trong số n dòng tiếp theo: mã của bạn thứ i

Ví dụ

Team.inp	Output	
8	3	8 bạn:
ABBBAA	1 4 5	Bạn số 1 có mã ABBBAA
CDCDCD	2 6	...
EEEEEE	3 7 8	Bạn số 8 có mã VIETVI
FAAAAF		3 đội:
GGGFFF		Đội 1: 1 4 5
DXXXXX		Đội 2: 2 6
VTVTVT		Đội 3: 3 7 8
VIETVI		

Algorithm

Ta tưởng tượng mỗi bạn trẻ là một đỉnh của đồ thị, hai bạn x và y có tập các chữ cái trong mã giao nhau (tức là chung nhau ít nhất một chữ cái) thì tạo thành một cạnh $x-y$ của đồ thị. Khi đó số nhóm chính là số TPLT của đồ thị.

- ☞ Đọc từng thành viên thứ i , so sánh với các thành viên j trong danh sách t . Nếu i và j giao nhau tại một chữ cái thì thực hiện Union.
- ☞ Duyệt lại mảng liên kết d để hiển thị các thành phần liên thông, tức là các thành viên trong mỗi đội.

Độ phức tạp: Với mỗi bạn $i = 1..n$ ta phải duyệt lại các bạn đọc trước i do đó ta cần thời gian n^2 . Thao tác Union cần thời gian n , vậy tổng cộng độ phức tạp là n^3 .

Program

```
// IT Teams
#include<bits/stdc++.h>
using namespace std;
const char * FN = "TEAM.INP";
```

158 CHUYÊN TIN 11: CÁC CHUYÊN ĐỀ CHỌN LỌC VỚI NGÔN NGỮ C++

```
typedef vector<int> VI;
typedef vector<string> VS;
VI d; // tham chieu
int Find(int x) {
    while (d[x] != x) x = d[x];
    return x;
}
int Union(int x, int y) {
    x = Find(x); y = Find(y);
    if (x == y) return 0;
    if (x < y) d[y] = x; else d[x] = y;
    return 1;
}
// x va y chung nhau it nhat 1 chu cai
bool Intersection(const string & x, const string & y) {
    for(auto c:x) if (y.find(c) != -1) return 1;
    return 0;
}
void Team() {
    ifstream f(FN);
    if(!f) {
        cout << "\n unable open input file " << FN;
        exit(0);
    }
    int n; // so doi
    int k; // so TPLT
    VS t; // DS ma so cac thanh vien
    f >> n;
    cout << n << " participants " << endl;
    for(int i = 0; i <= n; ++i) d.push_back(i);
    k = n;
    t.push_back("#"); // guard
    string x;
    f >> x;
    t.push_back(x); // the first participant
    k = n; // k: so TPL;
```

```

for(int i = 2; i <= n; ++i) {
    f >> x;
    for (int j = 1; j < i; ++j)
        if (Intersection(x, t[j])) k -= Union(i, j);
    t.push_back(x);
} // i
f.close();
// Dat lai tro truc tiep
for(int i = 1; i <= n; ++i) d[i] = Find(i);
cout << k << " teams" << endl;
int nn = 0; // team number
for(int i = 1; i <= n; ++i) {
    if (d[i] == i) {
        ++nn;
        cout << " Team no " << nn << ":";
        for(int j = i; j <= n; ++j)
            if (d[j] == i) cout << " " << j;
        cout << endl;
    }
} // for i
}

int main() {
    Team();
    cout << " T h e   E n d";
    return 0;
}

```

Output

Input:

8 participants

3 teams

Team no 1: 1 4 5

Team no 2: 2 6

Team no 3: 3 7 8

T h e E n d

Ants

Bi là cậu bé rất thích nuôi kiến vì Bi cho rằng kiến rất giỏi tìm đường đi ngắn nhất trong mê cung. Bi dùng đất nặn để xây dựng một mê cung hoành tráng gồm n lòng chảo và m đường hào nối các lòng chảo với nhau. Sau khi thả kiến vào một lòng chảo, Bi phát hiện ra rằng kiến không bao giờ bỏ đến một số lòng chảo. Bạn hãy cho biết Bi cần làm thêm ít nhất là bao nhiêu đường hào để kiến có thể xuất phát từ một lòng chảo bất kỳ tìm được đến một lòng chảo bất kỳ, nơi Bi đã đặt sẵn cục đường phèn.

Input text file "ANT.INP":

- Dòng đầu tiên hai số nguyên dương n và m ; $\max(n) = 2000$; $\max(m) = 5000$;
- Dòng thứ i trong số m dòng tiếp theo: hai số nguyên dương x và y thể hiện một đường hào nối trực tiếp lòng chảo x với lòng chảo y .

Output:

- Dòng đầu tiên: số nguyên k cho biết số đường hào tối thiểu cần xây thêm.
- Dòng thứ i trong số k dòng tiếp theo: hai số nguyên dương u và v thể hiện một đường hào cần làm thêm để nối lòng chảo u với lòng chảo v .

Ví dụ

ANT.INP	Output
8 5	2
2 5	1 3
3 7	1 6
8 6	
5 1	
4 3	

Algorithm

Xem bài Net.

Subgraph

Cho đồ thị liên thông vô hướng G gồm N đỉnh có số hiệu từ 1 đến N và M cạnh. Giữa hai đỉnh khác nhau của G có không quá một cạnh nối hai đỉnh đó. Cho số hiệu của hai đỉnh S và E trong đồ thị, $1 \leq S \leq E \leq N$. Xét đồ thị con $H(S, E)$ của G gồm các đỉnh có mã số từ S đến E và các cạnh (x, y) nối hai đỉnh nằm trọn trong khoảng từ S đến E : $S \leq x, y \leq E$.

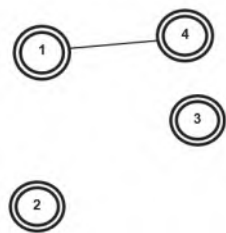
Với mỗi bộ ba giá trị S , E và V , $S \leq V \leq E$ cần hiển thị vùng liên thông chứa đỉnh V của đồ thị con $H(S, E)$.

Input: text file GRAPH.INP

- Dòng đầu tiên: 2 số $N\ M$ được ghi cách nhau.
- Dòng tiếp theo: M giá trị $x[1] \dots x[M]$ được ghi cách nhau.
- Dòng tiếp theo: M giá trị $y[1] \dots y[M]$ được ghi cách nhau.
($x[j], y[j]$) là cạnh thứ j của đồ thị liên thông G , $1 \leq j \leq M$.
- Dòng tiếp theo: số test Q
 Q dòng tiếp theo: mỗi dòng ba số S, E, V , $1 \leq S \leq V \leq E \leq N$ được ghi cách nhau.
Giới hạn: $1 \leq N \leq 200000$; $1 \leq M \leq 400000$; $1 \leq Q \leq 400000$.

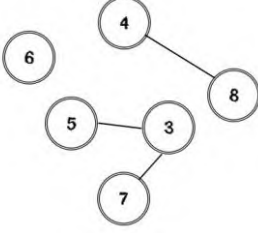
Ví dụ

GRAPH. INP	Đồ thị liên thông có $N = 8$ đỉnh, $M = 7$ cạnh: $(1,6), (1,5), (4,1), (4,8),$ $(3,5), (7,3), (5,2)$. $Q = 2$ test Test 1. $S = 1, E = 4, V = 4$: Hiển thị: 1 4. Test 2. $S = 3, E = 8, V = 7$: Hiển thị: 3 5 7.	
8 7 1 1 4 4 3 7 5 6 5 1 8 5 3 2 2 1 4 4 3 8 7		



Test 1: $S = 1, E = 4, V = 4$

$H(S, E)$ có ba miền liên thông.



Test 2: $S = 3, E = 8, V = 7$

$H(S, E)$ có ba miền liên thông

Đỉnh $V = 4$ trong miền liên thông 1 4. Đỉnh $V = 7$ trong miền liên thông 3 5 7

Algorithm

Thực hiện Find-Union để gán trị cho mảng tham chiếu d theo các cạnh (x,y) nằm trọn trong đồ thị con, Subgraph(S,E) nghĩa là thỏa mãn điều kiện $S \leq x, y \leq E$.

Để tránh việc duyệt mọi cạnh của đồ thị ta tổ chức một từ điển vt $\{x:s\}$ trong đó x là đỉnh của đồ thị, s là tập các đỉnh y liên thuộc đỉnh x , $y > x$.

Ví dụ trong đề bài cho ta:

- vt[1] = {4, 5, 6}
- vt[2] = {5}
- vt[3] = {5, 7}

162 CHUYÊN TIN 11: CÁC CHUYÊN ĐỀ CHỌN LỌC VỚI NGÔN NGỮ C++

vt[4] = {8}

vt[5] = vt[6] = vt[7] = vt[8] = {}

Hàm Less(s, e) cho ra tập các phần tử y trong tập s thỏa điều kiện lọc $y \leq e$.

Độ phức tạp

☞ Đọc dữ liệu: $O(NM)$

☞ Tính ConnectedZone(s,e,v): $O(NM)$

Program

```
// Subgraph
#include<bits/stdc++.h>
using namespace std;
const char * FN = "GRAPH.INP";
const char BL = 32;
typedef vector<int> VI;
typedef vector<VI> VVI;
int n; // so dinh
VVI ed; // ed[x] cac dinh y lien thuoc dinh x, x < y
const VI Empty; // vector rong
VI d; // mang tham chieu
// hien thi vector v kem chu thich msg
void Print(VI v, const char* msg = "") {
    cout << msg;
    for (auto x:v) cout << " " << x;
}
// hien thi vector<VI> v kem chu thich msg
void Print(VVI v, const char* msg = "") {
    cout << msg;
    for (int i = 0; i < v.size(); ++i) {
        if (v[i].empty()) continue;
        cout << "\n " << i << ":"; Print(v[i]);
    }
}
int Find(int x) {
    while (d[x] != x) x = d[x];
    return x;
}
```

```

int Union(int x, int y) {
    x = Find(x); y = Find(y);
    if (x == y) return 0;
    if (x < y) d[y] = x; else d[x] = y;
    return 1;
}
// cac dinh <= e
VI Less(VI v, int e) {
    VI w;
    for (auto z:v) if (z <= e) w.push_back(z);
    return w;
}
// TPLT chua dinh v
// of subgraph(s,e)
void ConnectedZone(int s, int e, int v) {
    d.clear();
    for(int i = 0; i <= n; ++i) d.push_back(i);
    for (int i = s; i <= e; ++i) {
        for (auto z:Less(ed[i], e)) Union(i, z);
    } // for i
    // dat lai tro truc tiep
    for (int i = 1; i <= n; ++i) d[i] = Find(i);
    cout << "\n Result:";
    for (int i = 1; i <= n; ++i) if (d[i] == d[v]) cout << " " << i;
}

void Run() {
    cout << " Input: ";
    ifstream f(FN);
    if(!f) {
        cout << "\n unable open input file " << FN;
        exit(0);
    }
    int m;
    f >> n >> m;
    cout << "\n " << n << " vertices " << m << " edges";
    ed.assign(n+1, Empty);
    VI x(m, 0);

```

```

// doc cac canh x
for (int i = 0; i < m; ++i) f >> x[i];
int y;
// doc cac canh y
// ghep dan vao ed[v] chua cac dinh y thoa v->y, v < y
for (int i = 0; i < m; ++i) {
    int v = x[i];
    f >> y; if (v > y) ed[y].push_back(v);
    else ed[v].push_back(y);
}
int q; // so test
f >> q;
int s, e, v;
cout << "\n " << q << "  tests";
for (int i = 1; i <= q; ++i) {
    f >> s >> e >> v;
    cout << "\n Test No " << i << ": "
        << " subgraph " << s << ".." << e
        << ". Connected zone containing " << v << " ?";
    ConnectedZone(s, e, v);
}
f.close();
}
int main() {
    Run();
    cout << endl << " T h e   E n d";
    return 0;
}

```

Output

Input:

8 vertices 7 edges

2 tests

Test No 1: subgraph 1..4. Connected zone containing 4 ?

Result: 1 4

Test No 2: subgraph 3..8. Connected zone containing 7 ?

Result: 3 5 7

T h e E n d

Werewolf

IOI Japan, 2018

Quận Ibaraki ở Nhật Bản có N thành phố và M con đường. Các thành phố được đánh số từ 0 đến $N-1$ theo thứ tự tăng dần về số lượng dân trong thành phố. Mỗi con đường nối một cặp hai thành phố khác nhau và có thể di chuyển theo cả hai hướng. Bạn có thể sử dụng một hoặc nhiều con đường này để di chuyển từ một thành phố bất kỳ đến một thành phố bất kỳ khác.

Bạn lên kế hoạch gồm Q chuyến đi, được đánh số từ 0 đến $Q-1$. Chuyến đi thứ i ($0 \leq i \leq Q-1$) sẽ di chuyển từ thành phố S_i đến thành phố E_i .

Bạn là người sói. Bạn có hai hình dáng: **Hình dáng người** và **hình dáng sói**. Tại thời điểm bắt đầu của mỗi chuyến đi, bạn ở hình dáng người. Kết thúc chuyến đi, bạn bắt buộc phải ở hình dáng sói. Trong chuyến đi, bạn phải **chuyển đổi** từ hình dáng người sang hình dáng sói đúng một lần. Bạn chỉ có thể chuyển đổi khi bạn đang ở một thành phố nào đó (kể cả S_i hoặc E_i).

Cuộc sống của người sói không dễ dàng. Bạn phải tránh ở hình dáng người khi ở các thành phố ít dân và tránh ở hình dáng sói khi ở các thành phố đông dân. Với mỗi chuyến đi i ($0 \leq i \leq Q-1$), có hai ngưỡng L_i và R_i ($0 \leq L_i \leq R_i \leq N-1$) chỉ ra các thành phố nào bạn phải tránh. Cụ thể là, bạn không được ở hình dáng người khi ở các thành phố $0, 1, \dots, L_i-1$, và không được ở hình dáng sói khi ở các thành phố $R_i+1, R_i+2, \dots, N-1$. Điều đó có nghĩa là trong chuyến đi thứ i , bạn chỉ được chuyển đổi hình dáng ở một trong các thành phố $L_i, L_i+1, \dots, R_i$.

Nhiệm vụ của bạn là xác định xem đối với mỗi chuyến đi có tồn tại hay không cách di chuyển từ thành phố S_i đến E_i thỏa mãn tất cả các điều kiện ở trên. Chuyến đi của bạn có thể có độ dài bất kỳ.

Understanding

Ta tinh giản đề như sau:

Cho đồ thị G vô hướng, liên thông gồm N đỉnh và M cạnh. Các đỉnh được mã số từ 0 đến $N-1$. Cho hai đỉnh giới hạn L và R , $0 \leq L \leq R \leq N$.

Xét hai đồ thị con H và W của G .

☞ H gồm các đỉnh từ L đến $N-1$ và các cạnh chứa các đỉnh đó.

☞ W gồm các đỉnh từ 0 đến R và các cạnh chứa các đỉnh đó.

Cho hai đỉnh $S \in H$ và $E \in W$.

Hãy cho biết có tồn tại hay không một đường đi gồm hai đoạn h và w thỏa mãn các điều kiện sau:

☞ h là đường đi trong đồ thị con H xuất phát từ S kết thúc tại đỉnh V .

☞ w là đường đi trong đồ thị con W xuất phát từ V kết thúc tại E ?

0	1	2	...	L - 1	L	L + 1	...	R - 1	R	R + 1	...	N - 1
					S ∈ H người							
E ∈ W sói												
					chuyển đổi người - sói							

Algorithm

Ý tưởng của thuật toán khá đơn giản: Ta cần tìm một đường đi

$$v = (S = v_1, \dots, v_i, \dots, v_j, \dots, v_k = E)$$

đáp ứng được các tiêu chí sau:

☞ $v_1 = S, v_k = E$: Xuất phát từ đỉnh S trong đồ thị con H , kết thúc tại đỉnh E trong đồ thị con W .

☞ $(v_1, \dots, v_i) \in H$: đoạn đường này nằm trọn trong H .

☞ $(v_j, \dots, v_k) \in W$: đoạn đường này nằm trọn trong W .

Từ đây suy ra rằng đoạn đường $(v_i, \dots, v_j)$ phải nằm trong phần giao của hai đồ thị con H và W , và đây cũng chính là nơi chuyển đổi từ lột người sang lột sói.

Bài toán Werewolf sẽ có nghiệm khi và chỉ khi giao của hai tập đỉnh $H(S)$ và $W(E)$ không rỗng, trong đó $H(S)$ là miền liên thông trong đồ thị con H chứa đỉnh S , $W(E)$ là miền liên thông trong đồ thị con W chứa đỉnh E .

Thật vậy, gọi V là điểm trong $H(S) \cap W(E)$. Do $V \in H(S)$, $H(S)$ là miền liên thông trong đồ thị con H và chứa đỉnh S nên từ S có thể di chuyển trong H để đến V , nghĩa là có đường đi trong H từ S đến V . Ta ký hiệu đường đi này là $h: S \rightarrow V$. Tương tự, $V \in W(E)$ khi và chỉ khi có đường đi trong đồ thị con W từ V đến E , ký hiệu là $w: V \rightarrow E$. Từ đó suy ra là có đường đi hai chặng từ S đến E như sau:

$S \rightarrow V \rightarrow E$. Cụ thể là:

Từ S trong H bạn di chuyển đến V trong lột Người: $(HS) \rightarrow (HV)$.

Tại V bạn đổi sang lột sói: $(HV)/(WV)$.

Rồi bạn di chuyển tiếp trong lột sói từ V đến E : $(WV) \rightarrow (WE)$.

Lưu ý rằng, do V là giao của $H(S)$ và $W(E)$ nên $L \leq V \leq R$, nghĩa là tại V bạn có thể chuyển đổi lột từ người sang lột sói. Vì lý do này nên ta gọi các đỉnh trong đoạn $[L;R]$ là vùng “trung lập”.

Ngược lại, khi $H(S) \cap W(E) = \emptyset$ thì không có đỉnh trung lập nào kết nối giữa hai vùng liên thông $H(S)$ và $W(E)$ nên không thể có đường đi $S \rightarrow ? \rightarrow E$.

Thuật toán được phác thảo như sau:

Bước 1. Xác định hai đồ thị con H và W . Độ phức tạp $O(R)$, $O(N-L)$

Bước 2. Xác định các miền liên thông chứa điểm xuất phát S của H , ký hiệu là $H(S)$.
 $O((N-L)N)$

Bước 3. Xác định các miền liên thông chứa điểm kết thúc E của W , ký hiệu là $W(E)$.
 $O(RN)$

Bước 4. Kết luận:

✎ Xác định giao T của $H(S)$ và $W(E)$.

✎ Nếu T là rỗng: trả lời vô nghiệm; ngược lại: trả lời có đường đi từ S tới E .

Input

Dữ liệu vào được ghi trong file text WEREWOLF.INP với các dòng như sau:

2 số N và M

M giá trị $x[0], \dots, x[M-1]$

M giá trị $y[0], \dots, y[M-1]$

Q test

Mỗi test được cho dưới dạng 4 số

S, E, L, R

Các dãy số $x[j]$ và $y[j]$ cho biết đường thứ j nối thành phố $x[j]$ với thành phố $y[j]$,
 $0 \leq j < M$.

Các số cùng dòng được ghi cách nhau.

Giới hạn

- $2 \leq N \leq 200\,000$
- $N - 1 \leq M \leq 400\,000$
- $1 \leq Q \leq 200\,000$
- Với mỗi $0 \leq j \leq M - 1$
 - $0 \leq X_j \leq N - 1$
 - $0 \leq Y_j \leq N - 1$
 - $X_j \neq Y_j$

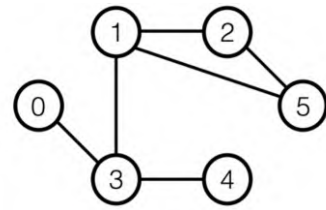
Chú ý

Trong bài này các chỉ số đỉnh được gán trị từ 0 đến $N-1$, các chỉ số cạnh được gán trị từ 0 đến $M-1$ và các test được gán thứ tự từ 0 đến $Q-1$.

Ví dụ

WEREWOLF.INP

6 6 Đồ thị gồm 6 đỉnh, 6 cạnh.
5 1 1 3 3 5 Các đỉnh được mã số từ 0 đến 5.
1 2 3 4 0 2 6 cạnh: (5,1), (1,2), (1,3), (3,4), (3,0), (5,2)
3 3 test
4 2 1 2 Test 0: S = 4, E = 2, L = 1, R = 2.
4 2 2 2 Test 1: S = 4, E = 2, L = 2, R = 2.
5 4 3 4 Test 2: S = 5, E = 4, L = 3, R = 4.



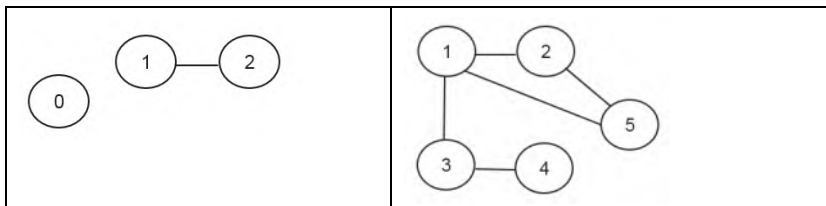
Giải trình Test 0

Test 0: S = 4, E = 2, L = 1, R = 2

①	①	②	③	④	⑤
W	W	W			
	H	H	H	H	H

H④ → H③ → H①W → W②

Test 0: S = 4, E = 2, L = 1, R = 2



Các vùng di chuyển của Sói

Các vùng di chuyển của Người

Test 0 chia đồ thị G thành hai đồ thị con H và W.

☞ H: các vùng dành cho Người:

Đỉnh: (1), (2), (3), (4), (5).

Cạnh: (1,2), (1,3), (1,5), (2,5), (3,4).

H có 1 miền liên thông: {1,2,3,4,5}.

Miền liên thông chứa điểm xuất phát S = 4: $H(4) = \{1,2,3,4,5\}$

☞ W: các vùng dành cho Sói.

Đỉnh: (0), (1), (2).

Cạnh: (1,2).

W có 2 miền liên thông: {0}, {1,2}.

Miền liên thông chứa đỉnh kết thúc $E = 2$: $W(2) = \{1, 2\}$

$$T = H(4) \cap W(2) = \{1, 2\}$$

Với Test này bạn di chuyển như sau:

- Xuất phát tại thành phố (H4): bạn ở hình dáng người (H4).
- Từ (H4) bạn di chuyển đến (H3): bạn ở hình dáng người (H4) $\rightarrow$ (H3).
- Từ (H3) bạn di chuyển đến (H1): bạn ở hình dáng người (H3) $\rightarrow$ (H1).
- Tại (H1) bạn chuyển sang lột sói: (H1)/(W1).
- Từ (W1) bạn di chuyển đến (W2) trong hình dáng sói: (W1) $\rightarrow$ (W2).

Sơ đồ di chuyển được viết như sau:

$$(H4) \rightarrow (H3) \rightarrow (H1)/(W1) \rightarrow (W2)$$

Bạn cũng có thể di chuyển đến (H2) rồi mới chuyển đổi lột:

$$(H4) \rightarrow (H3) \rightarrow (H1) \rightarrow (H2)/(W2)$$

Chú ý

Chỉ số trong bài này biến thiên trong khoảng $0..N$

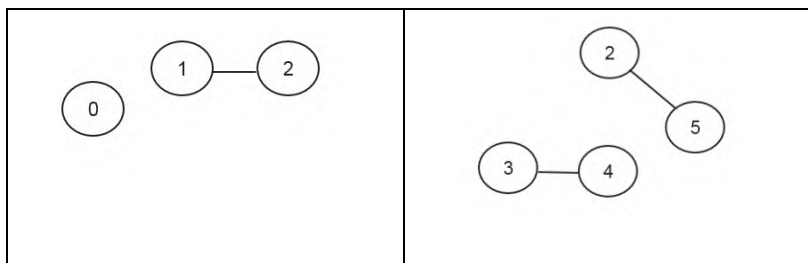
Giải trình Test 1

Test 1: $S = 4, E = 2, L = 2, R = 2$

①	①	②	③	④	⑤
W	W	W			
		H	H	H	H

Vô nghiệm

Test 1: $S = 4, E = 2, L = 2, R = 2$



Các vùng di chuyển của Sói

Các vùng di chuyển của Người

Test 1 chia đồ thị G thành hai đồ thị con H và W .

☞ H : các vùng dành cho Người:

Đỉnh: (2), (3), (4), (5).

Cạnh: (2,5), (3,4).

H có 2 miền liên thông: {2,5}, {3,4}.

Miền liên thông chứa điểm xuất phát $S = 4$: $H(4) = \{3,4\}$

☞ W: các vùng dành cho Sói.

Đỉnh: (0), (1), (2).

Cạnh: (1,2).

W có 2 miền liên thông: {0}, {1,2}.

Miền liên thông chứa đỉnh kết thúc $E = 2$: $W(2) = \{1,2\}$

$$T = H(4) \cap W(2) = \emptyset$$

Bạn xuất phát từ đỉnh $H(4)$ và chỉ có thể di chuyển đến $H(3)$ chứ không thể di chuyển đến $W(2)$. Vậy Test 1 vô nghiệm.

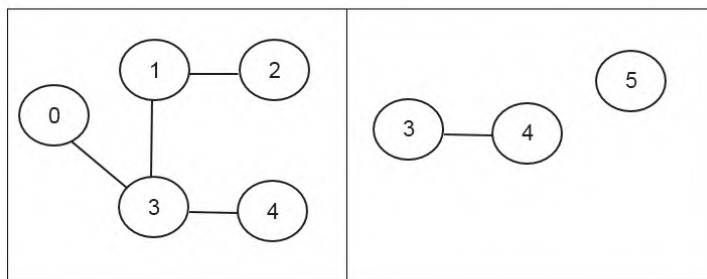
Giải trình Test 2

Test 2: $S = 5$, $E = 4$, $L = 3$, $R = 4$

①	②	③	④	⑤
W	W	W	W	
		H	H	H

Vô nghiệm

Test 2: $S = 5$, $E = 4$, $L = 3$, $R = 4$



Các vùng di chuyển của Sói

Các vùng di chuyển của Người

Test 2 chia đồ thị G thành hai đồ thị con H và W .

☞ H: các vùng dành cho Người:

Đỉnh: (3), (4), (5).

Cạnh: (3,4).

H có 2 miền liên thông: {5}, {3,4}.

Miền liên thông chứa điểm xuất phát $S = 5$: $H(5) = \{5\}$.

☞ W: các vùng dành cho Sói.

Đỉnh: (0), (1), (2), (3), (4).

Cạnh: (0,3), (1,2), (1,3), (3,4).

W có 1 miền liên thông: $\{0,1,2,3,4\}$.

Miền liên thông chứa đỉnh kết thúc $E = 4$: $W(4) = \{0,1,2,3,4\}$

$$T = H(5) \cap W(4) = \emptyset$$

Bạn xuất phát từ đỉnh cô lập $H(5)$ nên không thể di chuyển đến bất kỳ đỉnh nào khác. Vậy Test 2 vô nghiệm.

Cài đặt

Bài này dùng kỹ thuật Find-Union. Chi tiết kỹ thuật này bạn đọc có thể xem trong các bài đã trình bày ở phần trên, đặc biệt là bài Subgraph.

Ý tưởng của thuật toán khá đơn giản. Hai giới hạn trái L và phải R chia đồ thị G ban đầu thành hai đồ thị con: $0..L..R..N-1$

Đồ thị con H dành cho sự di chuyển của Người. H bao gồm các đỉnh $L..N-1$ và các cạnh (x,y) có hai đỉnh nằm trọn trong H: $L \leq x,y \leq N-1$.

Đồ thị con W dành cho sự di chuyển của Sói. W bao gồm các đỉnh $0..R$ và các cạnh (x,y) có hai đỉnh nằm trọn trong W: $0 \leq x,y \leq R$.

Trong đồ thị con H bạn dùng kỹ thuật Find-Union tách ra miền liên thông $H(S)$ chứa đỉnh xuất phát (S). Theo bài Subgraph, ta dùng hàm Connected Zone(L, N-1, S) với độ phức tạp $O(NM)$

Tương tự, trong đồ thị con W bạn dùng kỹ thuật Find-Union tách ra miền liên thông $W(E)$ chứa đỉnh kết thúc (E). Theo bài Subgraph, ta dùng hàm Connected Zone(0, R, E) với độ phức tạp $O(NM)$

Nếu H và W giao nhau thì có đường từ S đến E; ngược lại là không có đường. Tìm giao của hai tập đỉnh có độ phức tạp $O(N)$.

Program

```
// WEREWOLF
#include<bits/stdc++.h>
using namespace std;
const char * FN = "WEREWOLF.INP";
const char BL = 32;
typedef vector<int> VI;
typedef vector<VI> VVI;
int n; // so dinh
VVI ed; // ed[x] cac dinh y lien thuoc dinh x, x < y
```

172 CHUYỀN TIN 11: CÁC CHUYỀN ĐỀ CHỌN LỌC VỚI NGÔN NGỮ C++

```

const VI Empty; // vector rong
VI d; // mang tham chieu
// hien thi vector v kem chu thich msg
void Print(VI v, const char* msg = "") {
    cout << msg;    for (auto e:v) cout << " " << e;
}
// hien thi vector<VI> v kem chu thich msg
void Print(VVI v, const char* msg = "") {
    cout << msg;
    for (int i = 0; i < v.size(); ++i) {
        if (v[i].empty()) continue;
        cout << "\n " << i << ":"; Print(v[i]);
    }
}
int Find(int x) {
    while (d[x] != x) x = d[x];
    return x;
}
int Union(int x, int y) {
    x = Find(x); y = Find(y);
    if (x == y) return 0;
    if (x < y) d[y] = x; else d[x] = y;
    return 1;
}
// cac dinh <= e
VI Less(VI v, int e) {
    VI w;
    for (auto z:v) if (z <= e) w.push_back(z);
    return w;
}
bool Intersection(const VI & h, const VI & w) {
    for (auto v:h)
        if (count(w.begin(), w.end(), v)) return 1;
    return 0;
}
// TPLT chua dinh v
// of subgraph(s,e)

```

```

VI ConnectedZone(int s, int e, int v) {
    d.clear();
    for(int i = 0; i <= n; ++i) d.push_back(i);
    for (int i = s; i <= e; ++i) {
        for(auto w:Less(ed[i], e)) Union(i, w);
    } // for i
    // dat lai tro truc tiep
    for (int i = 0; i < n; ++i) d[i] = Find(i);
    VI out;
    // cout << "\n Result:";
    for (int i = 0; i < n; ++i)
        if (d[i] == d[v]) out.push_back(i);
    return out;
}

void Run() {
    cout << " Input: ";
    ifstream f(FN);
    if(!f) {
        cout << "\n unable open input file " << FN;
        exit(0);
    }
    int m;
    f >> n >> m;
    cout << "\n " << n << " dinh " << m << " canh";
    ed.assign(n, Empty);
    VI x(m, 0);
    // doc cac canh x
    cout << "\n x: ";
    for (int i = 0; i < m; ++i) {
        f >> x[i]; cout << " " << x[i];
    }
    int y; // doc cac canh y
    cout << "\n y: ";
    // ghep dan vao ed[v] chua cac dinh y thoa v->y, v < y
    for (int i = 0; i < m; ++i) {
        int v = x[i];
        f >> y; cout << " " << y;
    }
}

```

174 CHUYÊN TIN 11: CÁC CHUYÊN ĐỀ CHỌN LỌC VỚI NGÔN NGỮ C++

```
        if (v > y) ed[y].push_back(v); else ed[v].push_back(y);
    }
    int q; // xu ly cac test
    f >> q;
    int S, E, L, R, v;
    cout << "\n " << q << " tests";
    for (int i = 0; i < q; ++i) {
        f >> S >> E >> L >> R;
        cout << "\n Test No " << i << ": "
            << S << BL << E << BL << L << BL << R;
        VI H = ConnectedZone(L, n-1, S);
        VI W = ConnectedZone(0, R, E);
        Print(H, "\n H:"); Print(W, "\n W:");
        if (Intersection(H, W)) cout << "\n YES";
        else cout << "\n NO";
    }
    f.close();
}

int main() {
    Run();
    cout << endl << " T h e   E n d";
    return 0;
}
```

Output

Input:

6 dinh 6 canh

x: 1 1 1 3 0 2

y: 5 2 3 4 3 5

Edges at vertex:

0: 3

1: 5 2 3

2: 5

3: 4

3 tests

Test No 0: 4 2 1 2

H: 1 2 3 4 5

W: 1 2

YES

Test No 1: 4 2 2 2

H: 3 4

W: 1 2

NO

Test No 2: 5 4 3 4

H: 5

W: 0 1 2 3 4

NO

T h e E n d

Bình luận

Bạn có thể dùng set để lưu các tập đỉnh như phương án dưới đây:

Program

```
// WEREWOLF using set
#include<bits/stdc++.h>
using namespace std;
const char * FN = "WEREWOLF.INP";
const char BL = 32;
typedef vector<int> VI;
typedef set<int> SI;
typedef vector<SI> VSI;
int n; // so dinh
VSI ed; // ed[x] cac dinh y lien thuoc dinh x, x < y
const SI Empty; // vector rong
VI d; // mang tham chieu
int Find(int x) {
    while (d[x] != x) x = d[x];
    return x;
}
int Union(int x, int y) {
    x = Find(x); y = Find(y);
    if (x == y) return 0;
    if (x < y) d[y] = x; else d[x] = y;
    return 1;
}
// cac dinh <= e
SI Less(SI v, int e) {
```

```

    SI w;
    for (auto z:v) if (z <= e) w.insert(z);
    return w;
}
SI ConnectedZone(int s, int e, int v) {
    d.clear();
    for(int i = 0; i <= n; ++i) d.push_back(i);
    for (int i = s; i <= e; ++i)
        for(auto w:Less(ed[i], e)) Union(i, w);
    // dat lai tro truc tiep
    for (int i = 0; i < n; ++i) d[i] = Find(i);
    SI out;
    for (int i = 0; i < n; ++i)
        if (d[i] == d[v]) out.insert(i);
    return out;
}
bool Intersection(const SI & h, const SI & w) {
    for (auto v:h) if (count(w.begin(), w.end(), v)) return 1;
    return 0;
}
void Run() {
    cout << " Input: ";
    ifstream f(FN);
    if(!f) {
        cout << "\n unable open input file " << FN;
        exit(0);
    }
    int m;
    f >> n >> m;
    cout << "\n " << n << " dinh " << m << " canh";
    ed.assign(n, Empty);
    VI x(m, 0);
    // doc cac canh x
    cout << "\n x: ";
    for (int i = 0; i < m; ++i) {
        f >> x[i]; cout << " " << x[i];
    }
}

```

```

int y; // doc cac canh y
cout << "\n y: ";
// ghep dan vao ed[v] chua cac dinh y thoa v->y, v < y
for (int i = 0; i < m; ++i) {
    int v = x[i];
    f >> y; cout << " " << y;
    if (v > y) ed[y].insert(v); else ed[v].insert(y);
}
int q; // xu ly cac test
f >> q;
int S, E, L, R, v;
cout << "\n " << q << " tests";
for (int i = 0; i < q; ++i) {
    f >> S >> E >> L >> R;
    cout << "\n Test No " << i << ": "
        << S << BL << E << BL << L << BL << R;
    SI H = ConnectedZone(L, n-1, S);
    SI W = ConnectedZone(0, R, E);
    if (Intersection(H,W)) cout << "\n YES";
    else cout << "\n NO";
}
f.close();
}
int main() {
    Run();
    cout << endl << " T h e   E n d";
    return 0;
}

```


Chương 3

ADVANCED SORTING

Merge Sort

Cho hai dãy số sắp tăng a và b. Trộn hai dãy này để tạo ra một dãy c sắp tăng.

Ví dụ

```
a = [-3, 0, 1, 1, 4, 17]
b = [1, 5, 5, 9, 20]
c = [-3, 0, 1, 1, 1, 4, 5, 5, 9, 17, 20]
```

Algorithm

Nếu ta ghép a và b thành một dãy c, sau đó sort thì độ phức tạp là $O(n\log(n))$ với n là tổng chiều dài của a và b.

Nếu trộn trực tiếp a và b theo thuật toán cân bằng dưới đây thì chỉ tốn $O(n)$.

Xét các cặp hai phần tử $a[i]$ và $b[j]$:

Nếu $a[i] \leq b[j]$ thì nạp $a[i]$ vào dãy kết quả c và chọn $a[i]$ tiếp theo

Nếu $a[i] > b[j]$ thì nạp $b[j]$ vào dãy kết quả c và chọn $b[j]$ tiếp theo

Sau đó vét nốt những phần tử còn lại của a và b đưa vào c.

Program

```
// Merge Sort
#include<bits/stdc++.h>
using namespace std;
const char BL = 32;
typedef vector<int> VI;
// hien thi vector v kem chu thich msg
void Print(VI v, const char* msg = "") {
    cout << msg;
    for (auto e:v) cout << BL << e;
}
VI MergeSort(const VI & a, const VI & b) {
    VI c;
    int i = 0, j = 0;
```

```

while (i < a.size() && j < b.size())
    if (a[i] <= b[j]) c.push_back(a[i++]);
    else c.push_back(b[j++]);
for (; i < a.size(); ++i) c.push_back(a[i]);
for (; j < b.size(); ++j) c.push_back(b[j]);
return c;
}

void Demo() {
    int aa[] = {-3, 0, 1, 1, 4, 17}; // array aa
    int n = sizeof(aa) / sizeof(int);
    VI a(aa, aa+n); // vector a
    Print(a, "\n a:");
    int ab[] = {1, 5, 5, 9, 20}; // array ab
    int m = sizeof(ab) / sizeof(int);
    VI b(ab, ab+m); // vector b
    Print(b, "\n b:");
    VI c = MergeSort(a, b);
    Print(c, "\n c:");
}

int main() {
    Demo();
    cout << endl << " T h e   E n d";
    return 0;
}

```

Output

```

a: -3 0 1 1 4 17
b: 1 5 5 9 20
c: -3 0 1 1 1 4 5 5 9 17 20
T h e   E n d

```

Cách sinh test

Sinh ngẫu nhiên dãy số sắp tăng:

☞ Sinh ngẫu nhiên số thứ nhất $a[0]$

☞ Từ số thứ hai trở đi = số trước + (một số ngẫu nhiên ≥ 0)

```
srand(time(NULL)); // Init random
```

```
VI Gen(int n) {
```

```
    int r = 100;
```

```

VI a(n);
a[0] = (rand() % r) - r ; // the first number
for(int i = 1; i < n; ++i) a[i] = a[i-1] + (rand() % r);
return a;
}

```

Topo sort

Để hoàn thiện một dự án người ta cần thực hiện n công đoạn (CD) mã số $1..n$. Mỗi CD phải được thực hiện trước một số CD khác. Ví dụ, để xây xong ngôi nhà ba tầng ta phải thực hiện các CD sau:

Đào móng

Xây tầng 1

Đổ trần tầng 1

Xây tầng 2

Đổ trần tầng 2

Xây tầng 3

Đổ trần tầng 3

Lát nền

Người ta mô tả

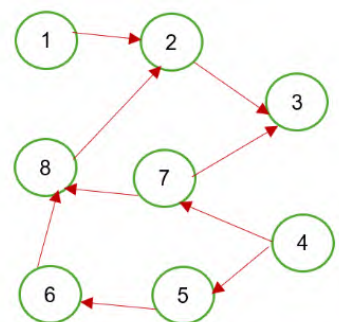
Đào móng < Xây tầng 1

để chỉ rõ CD Đào móng phải được thực hiện trước CD Xây tầng 1.

Biết các quan hệ trước sau của các CD trong một công trình / dự án. Hãy chỉ ra một trật tự thực hiện các CD để hoàn thành công trình.

Ví dụ

PROJECT . INP	8 công đoạn, 5 quan hệ trước sau:
8 5	CD 1 và 8 phải làm trước CD 2
1 8 < 2	CD 2 và 7 phải làm trước CD 3
2 7 < 3	CD 4 phải làm trước CD 5 và 7
4 < 5 7	CD 5 phải làm trước CD 6
5 < 6	CD 6 và 7 phải làm trước CD 8
6 7 < 8	



Algorithm

Xây dựng đồ thị có hướng: mỗi đỉnh là một CD, cung (x,y) cho biết CD x phải được thực hiện trước CD y .

Ký hiệu $\deg(x)$ là *bậc vào* của đỉnh x , tức là số cung *đi đến* đỉnh x . Ví dụ trên cho ta:

	1	2	3	4	5	6	7	8
deg	0	2	2	0	1	1	1	2

bậc của các đỉnh

Dễ thấy là các *đỉnh trực* là những đỉnh có bậc $\deg = 0$ cần được thực hiện trước. Sau khi chọn đỉnh trực t , ta đưa t ra kết quả rồi giảm bậc của các đỉnh kề sau đỉnh t 1 đơn vị. Bước tiếp theo sẽ tìm đỉnh trực tiếp theo và lặp lại thao tác giảm bậc.

Nếu tại bước nào đó không tìm được đỉnh trực tức là đồ thị có chu trình: một dãy các đỉnh tạo thành một đường đi khép kín, bài toán vô nghiệm.

Nếu tại mọi bước lặp đều có đỉnh trực thì bài toán có nghiệm là dãy liên tiếp các đỉnh trực tìm được tại mỗi bước lặp.

Nếu chiều dài của dãy kết quả là n thì thuật toán sắp topo thành công, ngược lại là thất bại.

Bài toán có thể có nhiều nghiệm, vì tại mỗi bước lặp có thể xuất hiện nhiều đỉnh trực. Nếu ta quy định chọn đỉnh trực có số hiệu nhỏ nhất thì nghiệm là duy nhất. Ví dụ trên cho ta nghiệm: 1 4 5 6 7 8 2 3.

Sau mỗi bước chọn đỉnh trực t , bạn nhớ đánh dấu $\deg[t] = N+1$ để tránh chọn lại lần sau.

Minh họa

	1	2	3	4	5	6	7	8	<i>đỉnh trực t</i>
deg	0	<u>2</u>	2	0	1	1	1	2	1 vì $1 \rightarrow 2$ nên --deg[2]
deg	x	1	2	0	<u>1</u>	1	<u>1</u>	2	4 vì $4 \rightarrow 5, 7$ nên --deg[5], --deg[7]
deg	x	1	2	x	0	<u>1</u>	0	2	5 vì $5 \rightarrow 6$ nên --deg[6]
deg	x	1	2	x	x	0	0	<u>2</u>	6 vì $6 \rightarrow 8$ nên --deg[8]
deg	x	1	<u>2</u>	x	x	x	0	<u>1</u>	7 vì $7 \rightarrow 3, 8$ nên deg[3], deg[8]
deg	x	<u>1</u>	1	x	x	x	x	0	8 vì $8 \rightarrow 2$ nên --deg[2]
deg	x	0	<u>1</u>	x	x	x	x	x	2 vì $2 \rightarrow 3$ nên --deg[3]
deg	x	x	0	x	x	x	x	x	3 Hoàn thành

Topo Sort: 1 4 5 6 7 8 2 3

Độ phức tạp

Thuật toán thực hiện lặp n lần, mỗi lần tìm một đỉnh trực với độ phức tạp $O(n)$. Giả sử mỗi lần xử lý đỉnh trực ta chỉ giảm bậc của một phần tử thì độ phức tạp chung sẽ là $O(n^2)$.

Chú thích

Thuật toán có tên là *topo sort*. Thuật toán này được vận dụng trong các bài toán quy hoạch, lập lịch hoạt động của các tác tử và các tiến trình trong hệ điều hành. Ngoài ra có thể dùng thuật toán để xác định chu trình trong lý thuyết đồ thị và trong xử lý tương tranh.

Program

```
// Project (topo sort)
#include<bits/stdc++.h>
using namespace std;
const char * FN = "PROJECT.INP";
const char BL = 32;
typedef vector<int> VI;
typedef set<int> SI;
typedef vector<SI> VSI;
int n; // number of works
VSI bf; // work must be done before all works in bf[x]
const SI Empty; // empty vector
VI deg; // vertex degreeh
// get set of numbers in string s
SI GetNum(string s, int limit = 0) {
    SI out;
    stringstream ss(s); // insert s to stringstream ss
    while (true) {
        int x = limit; // set x to limit value
        ss >> x; // get next int x from ss
        if (x == limit) break; // break if x == limit
        out.insert(x); // OK, take x
    }
    return out;
}
/*-----
    read data, compute deg[i]
    and bf[i]: works must do after work i
    for each i
-----*/
void Input() {
    cout << " Input: ";
    ifstream f(FN);
```

```

if(!f) {
    cout << "\n unable open input file " << FN;
    exit(0);
}
int m; // lines
f >> n >> m;
bf.assign(n+1, Empty);
deg.assign(n+1, 0);
string s;
cout << "\n " << n << " works " << m << " lines";
getline(f, s); // new line
for(int i = 1; i <= m; ++i) {
    getline(f, s); // f >> s;
    cout << "\n " << s;
    int j;
    for(j = 0; j < s.length(); ++j)
        if (s[j] == '<') break;
    SI ls = GetNum(s.substr(0, j)); // left side
    SI rs = GetNum(s.substr(j+1)); // right side
    for(auto x:ls)
        for(auto y:rs) {
            bf[x].insert(y);
            ++deg[y];
        }
} // for i
f.close();
// show data
for(int i = 1; i <= n; ++i) {
    cout << "\n work " << i << " must do before: ";
    for(auto x:bf[i]) cout << " " << x;
}
cout << "\n deg: ";
for(int i = 1; i <= n; ++i)
    cout << " " << deg[i];
}
// dinh bac 0 trong deg
int ZeroDeg() {

```

184 CHUYÊN TIN 11: CÁC CHUYÊN ĐỀ CHỌN LỌC VỚI NGÔN NGỮ C++

```
    for(int i = 1; i <= n; ++i)
        if (deg[i] == 0) {
            deg[i] = n+1; // danh dau
            return i;
        }
    return 0;
}

VI Topo() {
    VI t; // topo
    for(int i = 1; i <= n; ++i) {
        int z = ZeroDeg();
        if(z == 0) return t; // cycle
        t.push_back(z);
        for(auto y:bf[z]) --deg[y];
    }
    return t;
}

void Run() {
    Input();
    VI t = Topo();
    if(t.size() < n) cout << "\n Get a cycle: no solution. ";
    else {
        cout << "\n Get a topo: ";
        for(auto x:t) cout << " " << x;
    }
}

int main() {
    Run();
    cout << endl << " T h e   E n d";
    return 0;
}
```

Output

Input:

8 works 5 lines

1 8 < 2

2 7 < 3

4 < 5 7

```

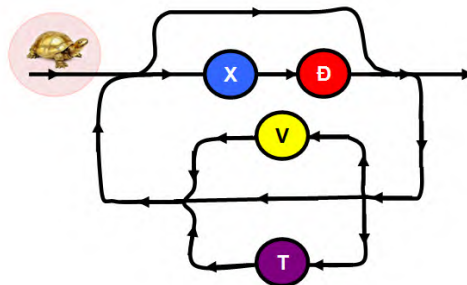
5 < 6
6 7 < 8
work 1 must do before: 2
work 2 must do before: 3
work 3 must do before:
work 4 must do before: 5 7
work 5 must do before: 6
work 6 must do before: 8
work 7 must do before: 3 8
work 8 must do before: 2
deg: 0 2 2 0 1 1 1 2
Get a topo: 1 4 5 6 7 8 2 3
T h e   E n d

```

Cách sinh test

- ✂ Vẽ trước một dãy công việc theo trật tự, ví dụ $5 \rightarrow 7 \rightarrow 1 \rightarrow 3 \rightarrow 2 \rightarrow 8 \rightarrow 6 \rightarrow 4$
- ✂ Thêm một vài mũi tên $\rightarrow$ cho dãy trên, ví dụ, $5 \rightarrow 3\ 4$, $3 \rightarrow 8\ 6 \dots$
- ✂ Muốn vô nghiệm: thêm mũi tên ngược chiều $\leftarrow$, ví dụ $6 \leftarrow 7$

Tortoise



Một chú Rùa bò vào mê cung và quanh quẩn một hồi rồi ra khỏi mê cung. Trong mê cung có 4 phòng Xanh (X), Đỏ (Đ), Tím (T) và Vàng (V). Mỗi khi bò qua phòng nào thì trên lưng Rùa sẽ nhận được thêm một chấm màu của phòng đó. Hãy cho biết khi ra khỏi mê cung trên lưng Rùa có thể có dãy chấm màu nào trong số các dãy cho trước?

TORTOISE . INP	OUTPUT	5 quan hệ trước sau >
5	YES	I cửa vào
I > OXVT	YES	O cửa ra
X > Đ	YES	X phòng xanh
Đ > OVTX	NO	Đ phòng đỏ
V > OXVT	YES	T phòng tím

T > OXVT	NO	V phòng vàng
6		6 test
XD _X D _V X _D T _X D		test 1: XD _X D _V X _D T _X D
V _T V _T T _X D		test 2: V _T V _T T _X D
X _D		...
X _D T _X		
V _V T _T		
T _V T _T X		

Algorithm

Đầu tiên ta mô tả mê cung theo phương thức trước-sau: từ điểm X Rùa có thể bò trực tiếp đến những điểm nào. Ta ký hiệu điểm I là *cửa vào* và O là *cửa ra* của mê cung, X là phòng xanh, D là phòng đỏ, T là phòng tím và V là phòng vàng... Ta thấy, theo hình vẽ:

Rùa phải xuất phát từ cửa vào I và kết thúc hành trình tại cửa ra O.

Từ điểm I Rùa có thể bò trực tiếp đến các điểm O, X, V và T. Ta viết $I > OXVT$.

Từ điểm X Rùa có thể bò trực tiếp đến điểm D. Ta viết $X > D$.

Từ điểm D Rùa có thể bò trực tiếp đến các điểm O, X, V và T. Ta viết $D > OXVT$.

Từ điểm V Rùa có thể bò trực tiếp đến các điểm O, X, V và T. Ta viết $V > OXVT$.

Từ điểm T Rùa có thể bò trực tiếp đến các điểm O, X, V và T. Ta viết $T > OXVT$.

Với mỗi dãy test s ta cho Rùa xuất phát từ cửa vào mê cung là I và xét từng ký tự c của s với mục đích: Rùa hiện đang ở điểm d, muốn bò đến điểm c.

Cuối cùng ta xét xem từ điểm cuối của test s Rùa có thể bò đến cửa ra O của mê cung hay không là hoàn tất một test.

Ta tổ chức sơ đồ mê cung dưới dạng một map<key, value> với trường key là một điểm (ký tự) k trong mê cung, trường value là một tập các điểm Rùa có thể bò trực tiếp từ k đến các điểm đó.

Program

```
// Tortoise
#include<bits/stdc++.h>
using namespace std;
const char * FN = "TORTOISE.INP";
const char BL = 32;
typedef set<char> SC; // set of char
typedef map<char, SC> MCSC;
MCSC maze; // so do di chuyen trong me cung
/*-----
```

```

    m[d] cho biet tu diem d
    rua co the bo den
    cac diem nao cua mecong
    -----*/
const SC EMPTY;
// Hien thi set s kem chu thich msg
void Print(SC s, const char*msg="") {
    cout << msg; for(auto c:s) cout << " " << c;
}
// Hien thi map n kem chu thich msg
void Print(MCSC maze, const char*msg="") {
    cout << msg;
    for(auto e:maze) {
        cout << "\n " << e.first;
        Print(e.second);
    }
}
void Test(string s) {
    char ch = 'I'; // xuat phat tu diem vao I
    for (int i = 0; i < s.length(); ++i) {
        // mong den duoc diem s[i]
        if (!maze[ch].count(s[i])) { cout << " : NO"; return; }
        ch = s[i]; // den duoc s[i]
    }
    // Rua co the ra khoi me cong
    if (maze[ch].count('O')) cout << " : YES"; else cout << " : NO";
}
void Run() {
    ifstream f(FN);
    if(!f) {
        cout << "\n unable open input file " << FN;
        exit(0);
    }
    int n;
    f >> n; // lines of descriptions
    cout << n << " descriptions";
    string L, s, R;

```

```

getline(f, s); // new line
maze.clear();
for(int i = 1; i <= n; ++i) {
    f >> L >> s >> R; // left > Right, s = >
    char ch = L[0]; maze[ch] = EMPTY;
    for(int j = 0; j < R.length(); ++j) maze[ch].insert(R[j]);
}
Print(maze);
int q;
f >> q;
cout << "\n " << q << " tests";
for(int i = 1; i <= q; ++i) {
    f >> s; cout << "\n Test No " << i << ": " << s;
    Test(s);
}
f.close();
}
int main() {
    Run();
    cout << endl << " T h e   E n d";
    return 0;
}

```

Output

```

5 descriptions
D :  O T V X
I :  O T V X
T :  O T V X
V :  O T V X
X :  D
6 tests
Test No 1: XDXDVXDTXD : YES
Test No 2: VTVTXD : YES
Test No 3: XD : YES
Test No 4: XDTX : NO
Test No 5: VVTT : YES
Test No 6: TVTTX : NO
T h e   E n d

```

Chú thích

Loại bài toán này cùng với các bài toán về dạng chuẩn Ba Lan Lukasiewicz trong xử lý các biểu thức tính toán được xếp vào lớp các bài toán xử lý ngôn ngữ bao gồm cả ngôn ngữ tự nhiên.

Cách sinh test

- ☞ Theo hình vẽ trong đề bài: thêm một vài điểm màu, ví dụ M: mặn, L: xanh lá, N: nâu,...
- ☞ Tự vẽ hình với các đường một chiều, đặt các điểm màu.

Floor Decoration

Người ta cần lát kín một nền nhà hình vuông cạnh dài $n = 2^k$ (k là một số tự nhiên trong khoảng 1..10) khuyết một ô vuông thoát nước góc dưới phải (ký hiệu là số 0) bằng những viên gạch màu hình thước thợ (chữ L) tạo bởi 3 ô vuông đơn vị như trong hình. Hai viên gạch kề cạnh nhau dù chỉ 1 đơn vị dài phải có màu khác nhau. Hãy cho biết một cách lát với số màu ít nhất.

Dữ liệu vào: số tự nhiên k .

Dữ liệu ra: tệp văn bản FLOOR.OUT: Dòng đầu tiên là hai số tự nhiên m biểu thị số viên gạch dạng L và c là số màu cần dùng cho việc lát nền. Tiếp đến là một phương án lát nền tìm được, trong đó mỗi viên gạch lát được tạo bởi ba chữ số giống nhau thể hiện màu của viên gạch đó. Các số trên mỗi dòng cách nhau qua dấu cách.

INPUT $k = 2$ ($n = 4$) FLOOR.OUT

1	1	3	3	5 3
1	2	2	3	1 1 3 3
3	2	1	1	1 2 2 3
3	3	1	0	3 2 1 1
				3 3 1 0

Algorithm

Lời giải sau đây của bạn Lê Hoàng Hải, lớp 10 chuyên Tin, Đại học Khoa học Tự nhiên, Đại học Quốc gia Hà Nội (năm 2002).

Để tính số viên gạch ta lấy diện tích hình trừ 1 ô thoát nước 0 rồi chia cho 3 là diện tích (số ô) của 1 viên gạch thước thợ:

$$\text{sogach} = (n*n - 1) \text{ div } 3; n = 2^k;$$

A	C				
D	B				

Về số màu, với $k = 1$, do đó $n = 2$ thì chỉ cần 1 viên gạch màu 1. Với mọi $k > 1$ ta sẽ trình bày một thuật toán cần *tối đa ba màu*.

Đầu tiên ta gọi thủ tục Init để tạo nền nhà dưới dạng ma trận a kích thước $n \times n$, tiếp đến ta khởi trị với hình vuông 2×2 nằm ở góc trên trái của nền nhà: $a[0][0] = a[0][1] = a[1][0] = 1$; $a[1][1] = 2$. Gọi hình được khởi trị là A. Mỗi bước tiếp theo ta thực hiện ba thao tác biến hình sau đây:

Tịnh tiến A đi xuống theo đường chéo chính thu được hình B (xem hàm Move).

Lật A sang phải (tức là lấy đối xứng gương qua cạnh dọc) để thu được hình C (xem hàm Right).

Lật A xuống dưới (tức là lấy đối xứng gương qua cạnh ngang) để thu được hình D (xem hàm Down).

Chú ý rằng khi lật ta cần thực hiện thêm phép hoán đổi trị 1 và 3 cho nhau.

Mỗi lần lặp như vậy ta sẽ thu được hình vuông có cạnh tăng gấp đôi hình trước. Khi $v = n$ ta gọi hàm Output để ghi ba mảnh D, A và C vào tệp kết quả.

1	1				
1	2				
		1	1		
		1	2		

Dịch Chéo

1	1	3	3		
1	2	2	3		
		1	1		
		1	2		

Lật Phải

1	1	3	3		
1	2	2	3		
3	2	1	1		
3	3	1	2		

Lật xuống

Bình luận

Thuật giải sử dụng hai phép biến hình cơ bản trong chương trình phổ thông là phép dời hình (tịnh tiến) và đối xứng qua trục. Việc hoán đổi trị 1 và 3 cho nhau là một ý tưởng thông minh. Mỗi ô trong bảng được điền đúng một lần do đó độ phức tạp tính toán của thuật toán là $O(n^2)$, trong khi các bài giải khác đều phải sử dụng các phép dò tìm để xác định màu tô và gọi đệ quy nên thường tốn kém về miền nhớ và thời gian hơn nhiều lần.

Program

Chương trình minh họa với nền nhà kích thước $k = 1..4$.

```
// Floor Decoration
#include<bits/stdc++.h>
```

```

using namespace std;
const char * GN = "FLOOR.OUT";
const char BL = 32;
typedef unsigned short US;
typedef vector<US> VUS;
typedef vector<VUS> WVUS;
int n; // canh nen nha
WVUS a;
inline void Go(const char * msg = " ? ") {
    cout << msg; fflush(stdin);
    if (cin.get() == '.') exit(0);
}
void Down(int m) {
    int m2 = m+m-1;
    for (int i = 0; i < m; ++i)
        for (int j = 0; j < m; ++j) a[m2-i][j] = 4-a[i][j];
}
void Right(int m) {
    int m2 = m+m-1;
    for (int i = 0; i < m; ++i)
        for (int j = 0; j < m; ++j) a[i][m2-j] = 4-a[i][j];
}
void Move(int m) {
    for (int i = 0; i < m; ++i)
        for (int j = 0; j < m; ++j) a[i+m][j+m] = a[i][j];
}
void Output() { // ghi file
    ofstream g(GN);
    g << (n*n-1) / 3 << BL; // so vien gach
    if(n == 2) g << 1; else g << 3;
    for(int i = 0; i < n; ++i) {
        g << endl;
        for (int j = 0; j < n; ++j) g << a[i][j] << BL;
    }
    g.close();
}

```

192 CHUYÊN TIN 11: CÁC CHUYÊN ĐỀ CHỌN LỌC VỚI NGÔN NGỮ C++

```
void FD(int k) {
    n = 1 << k; // n = 2^k
    cout << "\n k = " << k << "    n = " << n;
    VUS EMPTY;
    a.assign(n,EMPTY);
    for (int i = 0; i < n; ++i) a[i].assign(n,0);
    // vien dau tien tai Goc tren trai
    a[0][0] = a[0][1] = a[1][0] = 1; a[1][1] = 2;
    int m = 1;
    for(int i = 1; i < k; ++i) {
        m += m;
        Move(m); Right(m); Down(m);
    } // i
    a[n-1][n-1] = 0; // lo thoat nuoc
    Output();
}

int main() {
    FD(2);    // FD(3); FD(4);
    cout << endl << " T h e    E n d";
    return 0;
}
```

Output

Input k = 1 n = 2

Kết quả trong output file:

1 viên gạch 1 màu:

Sơ đồ lát:

1 1

1 0

?

Input k = 2 n = 4

Kết quả trong output file:

5 viên gạch 3 màu:

Sơ đồ lát:

1 1 3 3

1 2 2 3

3 2 1 1

3 3 1 0

?

Input k = 3 n = 8

Kết quả trong output file:

21 viên gạch 3 màu:

Sơ đồ lát:

1 1 3 3 1 1 3 3

1 2 2 3 1 2 2 3

3 2 1 1 3 3 2 1

3 3 1 2 2 3 1 1

1 1 3 2 1 1 3 3

1 2 3 3 1 2 2 3

3 2 2 1 3 2 1 1

3 3 1 1 3 3 1 0

?

Input k = 4 n = 16

Kết quả trong output file:

85 viên gạch 3 màu:

Sơ đồ lát:

1 1 3 3 1 1 3 3 1 1 3 3 1 1 3 3

1 2 2 3 1 2 2 3 1 2 2 3 1 2 2 3

. . .

?

T h e E n d

Chương 4

ỨNG DỤNG CÁC CHIẾN LƯỢC THIẾT KẾ THUẬT TOÁN NÂNG CAO

Flycam

Trong cuộc thi lập trình cho Flycam (thiết bị bay không người lái) thám hiểm vùng đất lạ, một nhóm bạn trẻ đã thắng cuộc với phần mềm độc đáo mang tên Rhythm (dáng điệu). Phần mềm này có khả năng xác định độ phẳng của một đoạn đường thẳng trên mặt đất chỉ bằng một lần bay duyệt từ đầu đến cuối đoạn đường và cho kết quả thể hiện qua ba đèn thông báo là D, A và E như sau:

	D	A	E	Kết quả	Vận hành
1	tắt	tắt	sáng	bằng phẳng	Flycam bay dọc theo đoạn đường,
2	tắt	sáng	tắt	tăng chặt	đo độ cao của $n \geq 2$ điểm trên mặt đất
3	tắt	sáng	sáng	tăng	theo chu kỳ đều đặn rồi so sánh
4	sáng	tắt	tắt	giảm chặt	độ cao tại điểm vừa đo với độ cao
5	sáng	tắt	sáng	giảm	của điểm đo sát trước.
6	sáng	sáng	tắt	mấp mô	
7	sáng	sáng	sáng	mấp mô	

Dáng điệu của đoạn đường được xác định như sau:

Gọi a là dãy cao độ với n điểm đo: $a[0], a[1], \dots, a[n-1]$

✧ Bằng phẳng: Nếu $a[0] = a[1] = \dots = a[n-1]$

✧ Tăng chặt: Nếu $a[0] < a[1] < \dots < a[n-1]$

✧ Tăng: Nếu $a[0] \leq a[1] \leq \dots < a[n-1]$

✧ Giảm chặt: Nếu $a[0] > a[1] > \dots > a[n-1]$

✧ Giảm: Nếu $a[0] \geq a[1] \geq \dots \geq a[n-1]$

✧ Mấp mô: Nếu không rơi vào các trường hợp trên

Theo ý bạn,

✧ Nhóm bạn trẻ chỉ dựa vào ba đèn D, A và C để suy luận ra sao?

✧ Hãy cung cấp một phương án thể hiện hàm Rhythm(a)?

Algorithm

✧ Với hai số x và y ta có ba trường hợp so sánh loại trừ nhau là $x = y$, $x < y$ và $x > y$.

✧ Gọi x là số đo tại bước i , $x = a[i]$, y là số đo tại bước sát trước bước i , $y = a[i-1]$.
Ta thấy:

- Đèn D ghi nhận trường hợp $y > x$, $D = 1$ (sáng), $D = 0$ (tắt)
- Đèn A ghi nhận trường hợp $y < x$, $A = 1$ (sáng), $A = 0$ (tắt)
- Đèn E ghi nhận trường hợp $y = x$, $E = 1$ (sáng), $E = 0$ (tắt)

Xét số nhị phân K tạo bởi ba bit D , A và E : $K = 4D+2A+E$, $k = 0..7$. Ta thấy không xảy ra trường hợp $k = 0$ nếu có chí ít hai lần đo

D	A	E	$K = 4D+2A+E$	Kết quả
0	0	1	1	bằng phẳng
0	1	0	2	tăng chặt
0	1	1	3	tăng
1	0	0	4	giảm chặt
1	0	1	5	giảm
1	1	0	6	mấp mô
1	1	1	7	mấp mô

Thuật toán khi đó sẽ như sau:

Dạng điệu của đoạn đường 5 thời điểm đo cao độ X: bật đèn X ($X \in \{D, A, E\}$)		Algorithm
	bằng phẳng: EEEE	Khởi trị các đèn $D = A = E =$ tắt Đo cao độ y tại điểm xuất phát cho mỗi lần: đo cao độ x if $x == y$: bật đèn E if $x < y$: bật đèn D if $x > y$: bật đèn A
	tăng chặt AAAA	
	tăng AEAE	
	giảm chặt DDDD	
	giảm EDED	
	mấp mô ADAD	

Program

Chương trình dùng hàm rev (lật vector) để sinh dữ liệu test với các dáng điệu khác nhau.

```
// Rhythm =1, >2, >=3, <4, <=5, ~0
#include<bits/stdc++.h>
using namespace std;
typedef vector<int> VI;
void Print(VI v, const char * msg = "") {
    cout << msg; for(auto x:v) cout << " " << x;
}
int Rhythm(VI a) {
    // Print(a, "Input:");
    int D = 0, A = 0, E = 0;
    for (int i = 1; i < a.size(); ++i)
        if (a[i] == a[i-1]) E = 1;
        else {
            if (a[i] > a[i-1]) A = 1;
            else D = 1;
        }
    return (4*D + 2*A + E) % 7 % 6;
}
VI Rev(VI a) {
    VI b;
    for (int i = a.size()-1; i >= 0; --i) b.push_back(a[i]);
    return b;
}
VI Concat(VI a, VI b) {
    VI c = a;
    for (int i = 0; i < b.size(); ++i) c.push_back(b[i]);
    return c;
}
int main() {
    int n = 10;
    VI b, t, g;
    // b là day bang, t: tang, g: giam
    b.assign(n, 3); // khoi tri day bang b
    Print(b, "\n Rhythm of b: ");
```

```

cout << " = " << Rhythm(b);
for (int i = 10; i <= 30; i += 3) t.push_back(i); // khoi tri doan tang chat
Print(t, "\n Rhythm of t: ");
cout << " = " << Rhythm(t);
g = Rev(t); // lat t thanh giam chat
Print(g, "\n Rhythm of g: ");
cout << " = " << Rhythm(g);
VI bt = Concat(b,t);
Print(bt, "\n Rhythm of bt: ");
cout << " = " << Rhythm(bt);
VI gb = Concat(g, b);
Print(gb, "\n Rhythm of gb: ");
cout << " = " << Rhythm(gb);
VI tg = Concat(t, g);
Print(tg, "\n Rhythm of tg: ");
cout << " = " << Rhythm(tg);
VI gt = Concat(g,t);
Print(gt, "\n Rhythm of gt: ");
cout << " = " << Rhythm(gt);
cout << endl << " T h e   E n d";
return 0;
}

```

Test

Để sinh dữ liệu test ta khởi tạo hai danh sách

- ☞ b: mọi phần tử đều bằng nhau và bằng c
- ☞ t: các phần tử tăng chặt tính từ c+1
- ☞ g là lật của danh sách t, g sẽ giảm chặt
- ☞ b+t sẽ là danh sách tăng
- ☞ g+b sẽ là danh sách giảm
- ☞ b+g và t+b sẽ là các danh sách mấp mô

Output

```

Rhythm of b:  3 3 3 3 3 3 3 3 3 3 = 1
Rhythm of t:  10 13 16 19 22 25 28 = 2
Rhythm of g:  28 25 22 19 16 13 10 = 4

```

Rhythm of bt: 3 3 3 3 3 3 3 3 3 10 13 16 19 22 25 28 = 3
 Rhythm of gb: 28 25 22 19 16 13 10 3 3 3 3 3 3 3 3 = 5
 Rhythm of tg: 10 13 16 19 22 25 28 28 25 22 19 16 13 10 = 0
 Rhythm of gt: 28 25 22 19 16 13 10 10 13 16 19 22 25 28 = 0
 T h e E n d

Philosophers

Dự tuyển Olympic Quốc tế

Chuyện xưa kể rằng n nhà hiền triết (NHT) viếng thăm đền Thông Tuệ để cầu xin sự thông thái. Trong đền có một phiến đá ghi số 0. Để đạt được lời thỉnh cầu, mỗi người phải thực hiện đủ n thao tác loại 1 và n thao tác loại 2 đan xen nhau, trong đó thao tác đầu tiên phải là loại 1.

Thao tác loại 1: Đọc	Thao tác loại 2: Ghi
✂ Vào đền	✂ Vào đền
✂ Đọc và ghi nhớ số trên phiến đá	✂ Lấy số nhớ trong đầu cộng thêm 1 và ghi kết quả lên phiến đá
✂ Ra khỏi đền	✂ Ra khỏi đền

Sau khi NHT ghi số mới trên phiến đá số cũ trên phiến đá tự động bị xóa và sau khi NHT thực hiện thao tác loại 2 (ghi) và ra khỏi đền thì NHT đó cũng quên luôn số đã nhớ.

Các NHT bàn bạc nhau để chọn ra một số linh thiêng là m. Hãy cho biết liệu các NHT có bố trí một lịch thực hiện để hoàn thành trọn vẹn lời thỉnh cầu với số cuối cùng hiển thị trên phiến đá là m?

Dữ liệu vào: 2 số n và m

Dữ liệu ra: Hiển thị một lịch thực hiện gồm một dãy tuần tự các dòng lệnh thuộc một trong hai dạng sau:

- ✂ Dạng thứ nhất gồm hai số tự nhiên ghi cách nhau: i t cho biết NHT i thực hiện thao tác t; $i \in \{1, 2, \dots, n\}$; $t \in \{1, 2\}$.
- ✂ Dạng thứ hai gồm 4 số tự nhiên ghi cách nhau: i t1 t2 k cho biết NHT i thực hiện k lần liên tiếp các thao tác t1 và t2 đan xen nhau; $i \in \{1, 2, \dots, n\}$; $t1, t2 \in \{1, 2\}$; $k > 0$.
- ✂ Nếu không có cách nào bố trí lịch thì ghi duy nhất một số 0.

Các số được ghi cách nhau.

Giới hạn

$$2 \leq n \leq 1000, 2 \leq m \leq 1000000$$

Ví dụ

Input	Output
3 4	1 1
	3 1 2 3
	1 2
	2 1
	1 1 2 2
	2 2
	2 1 2 2

3 NHT thăm đến với nhiệm vụ sinh ra kết quả trên

phiên đá (PD) là số 4. Lịch thực hiện như sau:

NHT 1: Đọc. PD = 0. Đầu(1) = 0.

NHT 3: (Đọc; Ghi) 3 lần. PD = 3.

NHT 1: Ghi. PD = Đầu(1)+1 = 0+1 = 1.

NHT 2: Đọc. Đầu(2) = 1, PD = 1.

NHT 1: (Đọc; Ghi) 2 lần. PD = 3.

NHT 2 Ghi. PD = Đầu(2)+1 = 1+1 = 2.

NHT 2 (Đọc;Ghi) 2 lần. PD = 2+2 = 4.

Chú thích: Đầu(i) - số nhớ trong đầu NHT thứ i.

Algorithm

Ta sẽ chỉ ra rằng với mọi $n \geq 2$ và mọi m trong khoảng $2..n^2$ luôn luôn có một lịch thỏa yêu cầu của đầu bài (ta gọi là *lịch hợp lệ*) để PD hiển thị giá trị cuối cùng là m .

Trước hết ta dễ thấy rằng nếu một NHT thực hiện liên tiếp k lần một cặp thao tác (TT) Đọc - Ghi (ĐG), tức là sau $2k$ dòng lệnh:

i 1

i 2

...

thì giá trị của PD được tăng thêm k đơn vị. Dãy $2k$ lệnh trên có thể viết gộp lại thành một lệnh 4 thành phần là:

i 1 2 k

Ta có hệ thức sau:

// PD = v: neu phien da hien chua so v

i 1 2 k // NHT i thuc hien k lan DG

// PD = v + k: thi PD se chua so v+k

Tiếp theo, lúc đầu ta có PD = 0. Giả sử ta cần xếp lịch cho n NHT thực hiện đủ n lần đọc, n lần ghi đan xen cho mỗi người để cuối cùng PD = m .

Ta chia m cho n để tìm thương k và dư r : $m = k*n + r$, $0 \leq r < n$.

Ta thấy, để đạt được giá trị m trên PD thì phải có k NHT thực hiện đầy đủ và liên tiếp n cặp TT ĐG, ngoài ra phải có ít nhất một NHT thực hiện thêm r cặp TT ĐG. Thật vậy:

Lúc đầu ta có PD = 0,

Sau khi k NHT mã số $v..v+k$ thực hiện n cặp TT ĐG, tức là sau khi thực hiện dãy lệnh:

for(int i = v; i ≤ v+k; ++i)

i 1 2 n

thì PD = $0 + k*n$.

Giả sử sau đó NHT u thực hiện r cặp TT ĐG

$$r \leq 2 \cdot n$$

thì $PĐ = k \cdot n + r = m$.

Do yêu cầu mỗi NHT buộc phải thực hiện n thao tác Đ và n thao tác G, một lẽ tự nhiên, ta phải chọn ra 2 NHT ghi nhận giá trị hiện có trên PĐ để khi cần sẽ trả lại giá trị đó (dĩ nhiên là phải cộng thêm 1) nhằm đảm bảo cho các thao tác cần thiết được thực hiện liên tục. Xét 3 trường hợp sau đây.

Trường hợp 1: $r = 0$, tức là $m = kn$. Ta cần k NHT thực hiện liên tiếp n cặp TT ĐG. Tuy nhiên, do những NHT khác cũng phải thực hiện đầy đủ n cặp TT ĐG cho mỗi người, nên ta chia các TT thành hai loại là các TT vô ích là những TT đến một thời điểm nào đó sẽ có một TT khác đặt lại giá trị cho PĐ. Các TT còn lại được gọi là TT có ích. Như vậy trường hợp này cần có k NHT thực hiện tổng cộng $m = kn$ cặp TT có ích và mọi TT còn lại sẽ là vô ích đối với NHT đó. Lịch khi đó sẽ như sau.

1 1 : NHT 1 Đọc và nhớ số 0 trên PĐ. Đầu(1) = 0.

i 1 2 n, $i = k+1..n$: các NHT còn lại (mang mã số $k+1..n$) ĐG n lần vô ích.

1 2 : NHT 1 ghi số 1 lên PĐ (có ích), $PĐ = \text{Đầu}(1)+1 = 0 + 1 = 1$.

1 1 2 n-1: NHT 1 ĐG nốt $n-1$ lần có ích, $PĐ = n$.

i 1 2 n, $i = 2..k$: các NHT $2..k$ ĐG n lần có ích, $PĐ = n + (k-1) \cdot n = k \cdot n = m$.

Trường hợp 2: $r \neq 0$ và $k > 0$. Ta có $m = kn + r$, $0 < r < n$, $k > 0$. Trường hợp này cần $n-1$ NHT thực hiện các TT vô ích, 1 NHT thực hiện r cặp TT ĐG có ích và cũng chính NHT đó phải thực hiện $(n-r)$ cặp TT vô ích. Ta yêu cầu 2 NHT 1 và 2 như sau.

1 1 : NHT 1 Đọc và nhớ số 0. Đầu(1) = 0.

i 1 2 n, $i = k+2..n$: các NHT $k+2$ đến n ĐG n lần vô ích.

1 2 : NHT 1 Ghi 1 lên PĐ. $PĐ = \text{Đầu}(1)+1 = 0 + 1 = 1$.

2 1 : NHT 2 Đọc và nhớ số 1. Đầu(2) = 1.

1 1 2 n-r : NHT 1 ĐG $n-r$ lần vô ích.

Tiếp đến là những TT có ích:

2 2 : NHT 2 Ghi số 2 lên PĐ, $PĐ = \text{Đầu}(2)+1 = 1 + 1 = 2$.

1 1 2 r-1 : NHT 1 ĐG nốt $r-1$ lần có ích, $PĐ = 2 + (r-1)$.

2 1 2 n-1 : NHT 2 ĐG nốt $n-1$ lần có ích, $PĐ = 2 + (r-1) + (n-1) = n+r$.

i 1 2 n, $i = 3..k+1$: các NHT $3..k+1$ ĐG n lần có ích, $PĐ = n+r + (k-1) \cdot n = k \cdot n + r = m$.

Trường hợp 3: $r \neq 0$ và $k = 0$, do đó $m = r \geq 2$. Trường hợp này cần $n-1$ NHT thực hiện các TT vô ích, 1 NHT thực hiện r cặp TT ĐG có ích và cũng chính NHT đó phải thực hiện $(n-r)$ cặp TT vô ích. Ta yêu cầu 2 NHT 1 và 2 như sau.

1 1 : NHT 1 Đọc và nhớ số 0 trên PĐ. Đầu(1) = 0.

i 1 2 n, $i = 3..n$: các NHT $3..n$ ĐG n lần vô ích.

2 1 2 n-1 : NHT 2 ĐG $n-1$ lần vô ích.

1 2 : NHT 1 Ghi số 1 lên PD. $PD = \text{Đầu}(1)+1 = 0+1 = 1$.

2 1 : NHT 2 Đọc số 1 trên PD. $\text{Đầu}(2) = 1$.

1 1 2 $n-r+1$: NHT 1 ĐG $n-r+1$ lần vô ích.

2 2 : NHT 2 Ghi số 2 lên PD. $PD = \text{Đầu}(2)+1 = 1+1 = 2$.

1 1 2 $r-2$: NHT 1 ĐG $r-2$ lần có ích, $PD = 2 + (r-2) = r$.

Phần dưới đây trình bày cấu trúc dữ liệu và các thủ tục đọc dữ liệu và xếp lịch. Hai hàm phụ trợ `Command2(i, tt)` và `Command4(i, tt1, tt2, k)` đảm nhận chức năng:

`Command2(i,t)`: NHT i thực hiện TT t (1: đọc hoặc 2: ghi)

`Command4(i,t1,t2,k)`: NHT i thực hiện liên tiếp hai TT $t1$ và $t2$ k lần

Để theo dõi tiến độ của lịch bạn có thể tạo thêm các biến sau đây:

Biến PD ghi nhận giá trị trên phiên đá trong đèn

List T1, T2 ghi nhận số lần thực hiện TT 1 và 2 của mỗi NHT

List D ghi nhận số trong đầu của mỗi NHT. Sau lệnh ghi ta có $D[i]=-1$ cho biết NHT này quên số vừa ghi.

Thủ tục Test chỉ được thực hiện khi bạn muốn cập nhật các giá trị PD, D, T1 và T2.

Biến Update quyết định chương trình có cập nhật các giá trị PD, D, T1 và T2.

Program

```
// Philosophers
#include<bits/stdc++.h>
using namespace std;
typedef vector<int> VI;
int PD;
VI D, T1, T2;
// PD: số ghi trên phiên đá
// D[i]: số lần NHT i đọc
// T1[i]: số lần NHT i thực hiện thao tác 1
// T2[i]: số lần NHT i thực hiện thao tác 2
bool Update = false; // điều khiển việc hiển thị
const char BL = 32;
inline void Go(const char * msg = " ? ") {
    cout << msg; fflush(stdin);
    if (cin.get() == '.') exit(0);
}
void Print(VI v, const char * msg = "") {
    cout << msg; for(auto x:v) cout << " " << x;
}
```


202 CHUYÊN TIN 11: CÁC CHUYÊN ĐỀ CHỌN LỌC VỚI NGÔN NGỮ C++

```
// NHT i thực hiện thao tác t k lần, t = 1|2
// D = -1: quên số đã đọc
void Command2(int i, int t) {
    // cout << endl << i << BL << t;
    if (Update) {
        if (t == 1) { D[i] = PD; T1[i] += 1; } // doc
        else { PD = D[i]+1; D[i] = -1; T2[i] += 1; } // ghi
    }
}

// NHT i thực hiện thao tác tt k lần
void Command4(int i, int t1, int t2, int k) {
    if (k < 1) return; // không làm gì
    // k >= 1
    // cout << endl << i << BL << t1 << BL << t2 << k;
    if (Update) {
        for (int j = 0; j < k; ++j) {
            D[i] = PD; T1[i] += 1; // doc
            PD = D[i] + 1; D[i] = -1; T2[i] += 1; // ghi
        }
    }
}

void Philosophers(int n, int m, bool ud = false) {
    Update = ud; // update Có hiện thì hay không
    cout << "\n Input: " << n << " NHT "
    << " Số cần hiện cuối cùng trên PD " << m;
    if (n < 2 || m < 2 || m > n*n) { cout << 0; return; }
    PD = 0; // số lúc đầu ghi trên PD
    D.assign(n+1, -1); // số trong đầu NHT
    T1.assign(n+1, 0); // T1[i] số lần NHT i thực hiện TT 1
    T2.assign(n+1, 0); // T2[i] số lần NHT i thực hiện TT 2
    // Print(D, "\n D: "); Print(T1, "\n T1: "); Print(T2, "\n T2: "); Go();
    int k = m / n, r = m % n;
    // k NHT có ích, r TT có ích
    if (r == 0) { // Case 1
        Command2(1,1); // PD=0, D[1]=0 ,TT[1]=1
        for (int i = k+1; i <= n; ++i) Command4(i, 1, 2, n);
        // PD = None T[i] = 2n, i=k+1..n
    }
}
```

```

    Command2(1,2); Command4(1,1,2,n-1);
    for (int i = 2; i <= k; ++i) Command4(i,1,2,n);
    return;
} // if r
// r > 0
if (k > 0) { // Case 2: r > 0, k > 0
    Command2(1,1);
    // Bỏ qua các NHT vô ích
    for (int i = k+2; i <= n; ++i) Command4(i, 1, 2, n);
    Command2(1, 2); Command2(2, 1);
    Command4(1, 1, 2, n-r); // Các TT vô ích của NHT 1
    // Đây là các TT có ích
    Command2(2, 2); Command4(1, 1, 2, r-1);
    Command4(2, 1, 2, n-1);
    for(int i = 3; i <= k+1; ++i) Command4(i, 1, 2, n);
    return;
}
// Case 3: k = 0, r > 0
Command2(1,1);
// Bỏ qua các NHT vô ích
for (int i = 3; i <= n; ++i) Command4(i,1,2,n);
// n-1 TT vô ích của NHT 2
Command4(2,1,2,n-1); Command2(1,2); Command2(2,1);
// các TT vô ích của NHT 1
Command4(1,1,2,n-r+1); Command2(2,2); Command4(1,1,2,r-2);
}

void Test(int n, int m) {
    Print(T1, "\n T1: ");
    Print(T2, "\n T2: ");
    if (!Update) return;
    // Test
    cout << "\n PD ? " << PD;
    if (m == PD) cout << " CORRECT.";
    else cout << " ERROR.";
    cout << "\n Thao tác Doc-Ghi ?";
    for (int i = 1; i <= n; ++i) {
        if (T1[i] != n) {cout << " ERROR."; return; }
    }
}

```

204 CHUYÊN TIN 11: CÁC CHUYÊN ĐỀ CHỌN LỌC VỚI NGÔN NGỮ C++

```
        if (T2[i] != n) {cout << " ERROR."; return; }
    }
    cout << " CORRECT.";
}

int main() {
    int n = 3;
    int nn = n*n;
    for (int m = 2; m <= nn; ++m) {
        Philosophers(n, m, true);
        Test(n, m); Go();
    }
    n = 5;
    nn = n*n;
    for (int m = 2; m <= nn; ++m) {
        Philosophers(n, m, true);
        Test(n, m); Go();
    }
    cout << endl << " T h e   E n d";
    return 0;
}
```

Output

Input: 3 NHT, Số cần hiện cuối cùng trên PD: 2

1 1

3 1 2 3

2 1 2 2

1 2

2 1

1 1 2 2

2 2

Số trên phiên đá: 2 2 ? : CORRECT

TT Đọc-Ghi: CORRECT

?

Input: 3 NHT, Số cần hiện cuối cùng trên PD: 3

1 1

2 1 2 3

3 1 2 3

1 2

1 1 2 2

Số trên phiên đá: 3 3 ? : CORRECT

TT Đọc-Ghi: CORRECT

?

Input: 3 NHT, Số cần hiện cuối cùng trên PD: 4

1 1

3 1 2 3

1 2

2 1

1 1 2 2

2 2

2 1 2 2

Số trên phiên đá: 4 4 ? : CORRECT

TT Đọc-Ghi: CORRECT

?

Input: 3 NHT, Số cần hiện cuối cùng trên PD: 5

1 1

3 1 2 3

1 2

2 1

1 1 2 1

2 2

1 1 2 1

2 1 2 2

Số trên phiên đá: 5 5 ? : CORRECT

TT Đọc-Ghi: CORRECT

?

Input: 3 NHT, Số cần hiện cuối cùng trên PD: 6

1 1

3 1 2 3

1 2

1 1 2 2

2 1 2 3

Số trên phiên đá: 6 6 ? : CORRECT

TT Đọc-Ghi: CORRECT

?

Input: 3 NHT, Số cần hiện cuối cùng trên PD: 7

```

1 1
1 2
2 1
1 1 2 2
2 2
2 1 2 2
3 1 2 3
. . .
T h e   E n d

```

Độ phức tạp

Thuật toán hiển thị tối đa $2n^2$ dòng lệnh

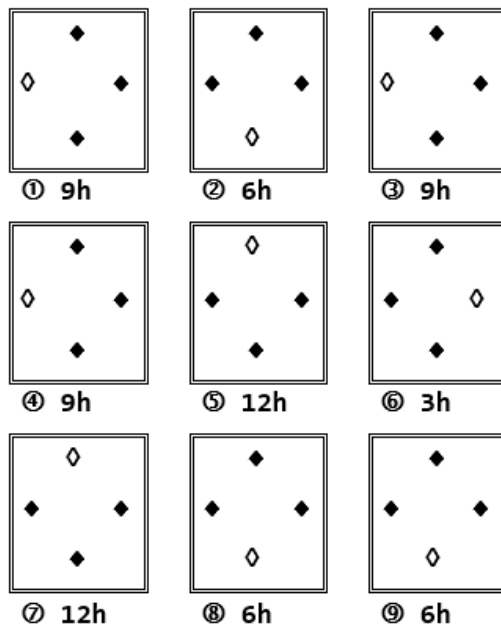
Bình luận

Bài toán *Philosophers* là một biến thể của bài toán lập lịch cho các *tiến trình tương tranh* như cấp phát tài nguyên (máy in, đọc ghi các file dữ liệu...). Nhiệm vụ của các phần mềm lập lịch tự động là phải quan tâm xử lý một cách tốt nhất các yêu cầu của các đối tác mà không làm tê liệt hệ thống.

Clocks

Olympic Quốc tế

Có 9 chiếc đồng hồ điện tử treo trên một bảng theo sơ đồ 3×3 . Các đồng hồ được mã số từ 1 đến 9 theo trật tự từ trái qua phải, từ hàng trên xuống hàng dưới. Mỗi đồng hồ có 4 điểm chỉ giờ ứng với các giờ 12, 3, 6 và 9. Giờ hiện trở của mỗi đồng hồ ứng với điểm sáng hình thoi $\diamond$. Để điều khiển các đồng hồ người ta sử dụng 9 phím với các chức năng được mô tả như trong hình vẽ. Khi nhấn vào một phím thì có 4 đồng hồ đồng



9 đồng hồ, $\diamond$ giờ đang trở

Sơ đồ bố trí
9 đồng hồ

①	②	③
④	⑤	⑥
⑦	⑧	⑨

9 phím
điều khiển

1	2	3
4	5	6
7	8	9

loạt nhảy điểm sáng đi 90° theo chiều kim đồng hồ để cộng thêm 3 giờ tính từ giờ hiện trở. Ví dụ, khi nhấn phím 1 thì 4 đồng hồ 1, 2, 4 và 5 sẽ được chỉnh giờ theo luật trên. Biết giờ hiện tại của các đồng hồ. Hãy xác định một dãy ngắn nhất các phím cần nhấn để các đồng hồ đồng loạt trở 12 giờ.

Dữ liệu vào: tệp văn bản CLOCKS.INP:

9 dòng đầu tiên, mỗi dòng 4 số, dòng thứ $i = 1..9$ cho biết khi bấm phím i thì 4 đồng hồ sẽ nhảy số.

Dòng thứ 10: số test

Mỗi test chiếm 3 dòng. Mỗi dòng mô tả hiện trạng ban đầu của 3 đồng hồ.

Các số trên cùng dòng cách nhau qua dấu cách.

Dữ liệu ra: Hiển thị trên màn hình dãy phím ngắn nhất cần nhấn.

Giả thiết rằng bài toán luôn có nghiệm..0

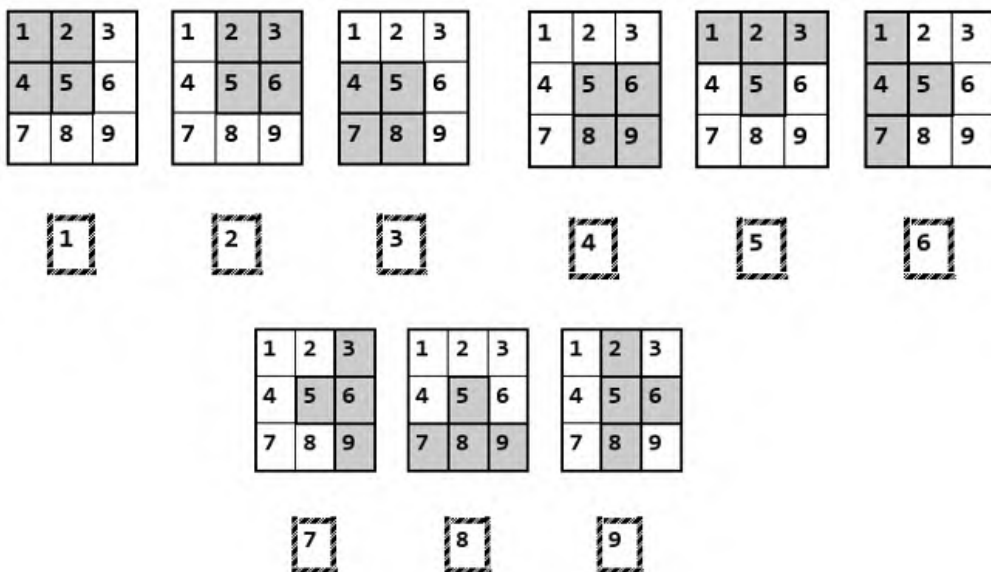
CLOCKS . INP	Ý nghĩa	CLOCKS.OUT
1 2 4 5	Nhấn phím 1	1244
2 3 5 6	Nhấn phím 2	137
4 5 7 8	Nhấn phím 3	22348
5 6 8 9	Nhấn phím 4	
1 2 3 5	Nhấn phím 5	
1 4 5 7	Nhấn phím 6	
3 5 6 9	Nhấn phím 7	
5 7 8 9	Nhấn phím 8	
2 5 6 8	Nhấn phím 9	
3	3 test	
9 6 9	đồng hồ 1 2 3	
9 12 3	đồng hồ 4 5 6	
12 6 6	đồng hồ 7 8 9	
9 9 9	đồng hồ 1 2 3	
6 3 9	đồng hồ 4 5 6	
9 9 9	đồng hồ 7 8 9	
12 6 6	đồng hồ 1 2 3	
9 9 3	đồng hồ 4 5 6	
6 3 6	đồng hồ 7 8 9	

Nhấn lần lượt các phím 1, 2, 4, 4

cả 9 đồng hồ đều đồng loạt trở 12 giờ.

Algorithm

Ta nhận thấy rằng, do các đồng hồ hoạt động độc lập với nhau, nên dãy phím cần nhấn là giao hoán, nghĩa là có thể liệt kê dãy phím đó theo trật tự tùy ý. Ví dụ, nhấn các phím 1, 2, 3 sẽ mang lại kết quả như khi nhấn dãy phím 2, 1, 3 hoặc theo bất kỳ hoán vị nào của 3 phím đó. Ngoài ra, nếu một phím được nhấn đến một bội lần của 4 thì điểm sáng trở giờ sẽ quay lại đúng vị trí ban đầu, do đó ta không nhấn một phím quá 3 lần. Từ hai nhận xét trên ta thấy rằng có thể dùng kỹ thuật vét cạn, cụ thể là xét các tổ hợp dãy phím $p[1..9]$, $p[i] = 0..3$ biểu thị số lần bấm phím i . Với mỗi tổ hợp p ta tính xem các đồng hồ được cập nhật ra sao. Nếu cả 9 đồng hồ đều nhảy về thời điểm 12 giờ thì ta chọn phương án có số lần bấm phím ít nhất trong số các tổ hợp ứng viên. Ví dụ, tổ hợp $x = 244$ cho biết phải bấm các phím điều khiển 4, 4 và 2.



Chức năng của các phím điều khiển

Tổ chức dữ liệu

☞ Danh sách 9 đồng hồ, $c[i]$ cho biết đồng hồ i hiện chỉ mấy giờ. Ta chuyển đổi giờ sang hệ đếm $\text{BASE} = 4$, cụ thể là $12 \rightarrow 0$, $3 \rightarrow 1$, $6 \rightarrow 2$, $9 \rightarrow 3$ theo công thức $h \rightarrow (h // 3) \% \text{BASE}$.

☞ vector $p[k] = d$ cho biết nếu bấm phím k trên bảng điều khiển thì các ĐH trong danh sách d sẽ dịch kim thêm 90° .

Program

```
// Clocks
#include<bits/stdc++.h>
using namespace std;
const char * FN = "CLOCKS.INP";
```

```

const char * GN = "CLOCKS.OUT";
typedef vector<int> VI;
VI c, d; // clocks
typedef vector<VI> VVI;
VVI p; // keys
// p[i]: bam key i thi 4 clocks trong p[i] chuyen 90 do
inline void Go(const char * msg = " ? ") {
    cout << msg; fflush(stdin);
    if (cin.get() == '.') exit(0);
}
void Print(VI v, const char * msg = "", int start = 0) {
    cout << msg;
    for(int i = start; i < v.size(); ++i) cout << " " << v[i];
}
void Print(VVI vv, const char * msg = "") {
    cout << msg;
    for(int i = 1; i <= 9; ++i) {
        cout << "\n Press key " << i << ": ";
        Print(vv[i], " ");
    }
}
// 9 dong ho deu chi vao 0 (12h)?
bool AllZero() {
    for (int i = 1; i <= 9; ++i)
        if (d[i] % 4 != 0) return false;
    return true;
}
// Kiem tra viec bam cac phim dieu khien
// la chu so cua x
// Vi du, x = 144 thi
// bam lan luot cac phim dieu khien 4, 4, 1
bool Check(int x) {
    d = c;
    // truong hop c toan 0: khong phai bam
    if (AllZero()) return x;
    // Tach x thanh cac chu so
    int i;

```



```

while (x) {
    i = x % 10; // bam phim i
    x /= 10;
    if (i == 0) continue;
    // i = 1..9: bam 4 phim
    for (int j = 0; j < 4; ++j) {
        ++d[p[i][j]];
    } // for
}
return AllZero();
}

void Run() {
    // Init
    c.assign(10, 0); // control tab
    VI e;
    e.assign(4, 0);
    p.assign(10, e);
    // read data
    ifstream f(FN);
    if(!f) {
        cout << "\n unable open input file " << FN;
        exit(0);
    }
    //line i: nhan key i thi 4 clocks dich vong 90 do
    string s;
    for (int i = 1; i <= 9; ++i) {
        getline(f, s);
        stringstream ss(s);
        for (int j = 0; j < 4; ++j)
            ss >> p[i][j];
    }
    Print(p, "\n Control Tab    Clocks");
    int q; // so test
    int x;
    f >> q;
    cout << endl << q << " tests.";
    getline(f, s);

```

```

ofstream g(GN); // output file
if(!g) {
    cout << "\n unable open output file " << GN;
    exit(0);
}
for (int i = 1; i <= q; ++i) {
    cout << "\n Test No " << i << ": ";
    // read 3 lines X 3 numbers set to c
    int k = 0; // clock thu k = 1..9
    for (int j = 1; j <= 3; ++j) {
        getline(f, s);
        stringstream ss(s); // doc gio cua 3 clocks
        for (int ii = 0; ii < 3; ++ii) {
            ss >> x;
            ++k; c[k] = (x / 3) % 4; // gio hien tro cua clock k
        } // ii
    } // j
    Print(c, "\n present times c: ", 1);
    for(x = 1; !Check(x); ++x); // vet can
    cout << "\n Result " << x;
}
f.close(); g.close();
}
int main() {
    Run();
    cout << endl << " T h e   E n d";
    return 0;
}

```

Output

Control Tab	Clocks
Press key 1:	1 2 4 5
Press key 2:	2 3 5 6
Press key 3:	4 5 7 8
Press key 4:	5 6 8 9
Press key 5:	1 2 3 5
Press key 6:	1 4 5 7
Press key 7:	3 5 6 9

212 CHUYÊN TIN 11: CÁC CHUYÊN ĐỀ CHỌN LỌC VỚI NGÔN NGỮ C++

```
Press key 8:  5 7 8 9
Press key 9:  2 5 6 8
3 tests.
Test No 1:
present times c:  3 2 3 3 0 1 0 2 2
Result 1244
Test No 2:
present times c:  3 3 3 2 1 3 3 3 3
Result 137
Test No 3:
present times c:  0 2 2 3 3 1 2 1 2
Result 22348
T h e   E n d
```

Unique Number

Olympic Baltics

Cho hai số nguyên tố cùng nhau k và m . Tập văn bản `UNIQUE.INP` chứa dãy số tự nhiên, mỗi số chiếm một dòng dài không quá 20 chữ số. Trong dãy này có duy nhất một số xuất hiện đúng k lần, các số còn lại đều xuất hiện đúng m lần. Hãy tìm số duy nhất đó.

Giới hạn

$1 \leq k < m \leq 100$. Số dòng trong input file tối đa là 5 triệu.

Ví dụ

UNIQUE.INP	OUTPUT	Số duy nhất xuất hiện
2 3	10203040	đúng 2 lần là 10203040.
1357		
10203040		Các số còn lại đều xuất hiện đúng 3 lần.
2008		
8010203040		
1357		
2008		
8010203040		
2008		
1357		
10203040		
8010203040		

Algorithm

Ta dựa vào một kiến thức có từ hàng ngàn năm trước, đó là biểu diễn số theo vị trí. Lần lượt đọc từng dòng vào một biến string, sau đó ghi vào một mảng a số lần xuất hiện của từng chữ số tại từng vị trí, $a[c][i]$ cho biết số lần xuất hiện của chữ số c tại cột i tính từ trái qua phải.

Như vậy mảng a có kích thước 10 dòng đủ chứa 10 chữ số 0..9 và 20 cột đủ chứa các chữ số của số dài nhất.

Sau khi đọc xong dữ liệu, ta thấy mọi phần tử của mảng a hoặc là chia hết cho m, tức là có dạng vm hoặc là không chia hết cho m, tức là có dạng $vm+k$, $(k,m)=1$. Nếu $a[c][i]$ không chia hết cho m thì c sẽ là chữ số xuất hiện tại vị trí i của số duy nhất cần tìm.

Độ phức tạp

$O(n)$, n là tổng chiều dài dữ liệu input (số chữ số).

Dữ liệu trong đề bài cho ta mảng a như sau:

	c	o	t									
0:	0	<u>8</u>	3	<u>5</u>	0	<u>5</u>	0	<u>5</u>	0	3	0	. . .
1:	<u>5</u>	0	3	0	0	0	0	0	0	0	0	. . .
2:	3	0	<u>2</u>	0	3	0	0	0	0	0	0	. . .
3:	0	3	0	0	<u>2</u>	0	3	0	0	0	0	. . .
4:	0	0	0	0	0	0	<u>2</u>	0	3	0	0	. . .
5:	0	0	3	0	0	0	0	0	0	0	0	. . .
6:	0	0	0	0	0	0	0	0	0	0	0	. . .
7:	0	0	0	3	0	0	0	0	0	0	0	. . .
8:	3	0	0	3	0	0	0	0	0	0	0	. . .
9:	0	0	0	0	0	0	0	0	0	0	0	. . .

Program

```
// Unique
#include<bits/stdc++.h>
using namespace std;
#define Int(c) (c)-'0'
const char * FN = "UNIQUE.INP";
const int N = 10;
const int M = 20;
const int MN = M*N;
inline void Go(const char * msg = " ? ") {
    cout << msg;    fflush(stdin);
```

214 CHUYÊN TIN 11: CÁC CHUYÊN ĐỀ CHỌN LỌC VỚI NGÔN NGỮ C++

```
    if (cin.get() == '.') exit(0);
}

void Run() {
    ifstream f(FN);
    if(!f) {
        cout << "\n unable open input file " << FN;
        exit(0);
    }
    int k, m;
    f >> k >> m; // (k, m) = 1 nguyên tố cùng nhau
    int a[N][M];
    memset(a, 0, MN*sizeof(int));
    string s;
    getline(f, s); // bỏ qua dòng chứa k m
    while (getline(f, s))
        for(int i = 0; i < s.length(); ++i)
            ++a[Int(s[i])][i]; // digit Int(s[i]) pos i
    f.close();
    // Duyệt lại a
    cout << endl << "Result: ";
    for (int pos = 0; pos < M; ++pos)
        for (int digit = 0; digit < N; ++digit)
            if (a[digit][pos] % m != 0) cout << digit;
}

int main() {
    Run();
    cout << endl << " T h e   E n d";
    return 0;
}
```

Output

```
Result: 10203040
T h e   E n d
```

Lập lịch cho hai máy

Có n phần mềm (PM) cần được test lần lượt trên hai máy tính theo trình tự nghiêm ngặt: máy A sau đó là máy B. Máy A chuyên test giao diện, máy B chuyên test dữ liệu. Biết thời gian PM i test trên máy A là $a[i]$ (phút), trên máy B là $b[i]$ (phút). Tại thời điểm

xuất phát 0, hai máy tính cùng được bật. Hãy xây dựng một lịch test hết n PM trên hai máy sao cho tổng thời gian hoạt động của các máy tính là ít nhất.

Ví dụ

TEST.INP	Ý nghĩa						
6	PM	1	2	3	4	5	6
3 2 2 1 1 2	A: a[i]	3	2	2	1	1	2
2 2 1 3 2 1	B: b[i]	2	2	1	3	2	1
	Lịch	? (4, 5, 2, 1, 3, 6)					

Output

4 5 2 1 6 3

Ý nghĩa của input

Có 6 PM.

PM 1 phải được test trên máy A trong 3 ph, trên máy B trong 2 ph.

PM 2 phải được test trên máy A trong 2 ph, trên máy B trong 2 ph.

PM 3 phải được test trên máy A trong 2 ph, trên máy B trong 1 ph.

PM 4 phải được test trên máy A trong 1 ph, trên máy B trong 3 ph.

PM 5 phải được test trên máy A trong 1 ph, trên máy B trong 2 ph.

PM 6 phải được test trên máy A trong 2 ph, trên máy B trong 1 ph.

Ý nghĩa của output

	Trục thời gian											
phút	1	2	3	4	5	6	7	8	9	10	11	12
A PM	4	5	2	2	1	1	1	3	3	6	6	
B PM		4	4	4	5	5	2	2	1	1	3	6

Kết luận

Máy A sẽ lần lượt test các PM theo trật tự 4, 5, 2, 1, 6, 3 và chuyển giao cho máy B theo trật tự đó.

Theo trật tự này thì hai máy sẽ không có thời gian ngắt quãng và cả làm việc sẽ dừng sau phút thứ 12. Dĩ nhiên máy B sẽ được khởi động sau phút thứ nhất.

Algorithm (Johnson)

Để ý rằng chọn thứ tự cho PM nào trên máy A để máy A làm việc liên tục là không quan trọng. Nhưng sau khi PM i được test trên máy A thì cần được chuyển qua máy B. Từ đó suy ra rằng mỗi PM i cần được xử lý nhanh nhất trên máy A. Vậy các PM trên

216 CHUYÊN TIN 11: CÁC CHUYÊN ĐỀ CHỌN LỌC VỚI NGÔN NGỮ C++

máy A cần được sắp tăng dần theo thời gian (TG) a. Máy B cần có đủ TG để chờ các PM chuyển từ máy A qua. Từ đó suy ra rằng các PM xử lý trên máy B phải được sắp giảm theo TG b. Ta có thuật toán sau:

- ✧ Split: Tách tập các PM P thành hai nhóm không giao nhau: PA gồm các PM có TG trên máy A nhỏ thua hoặc bằng TG trên máy B, $PA = [i: a[i] \leq b[i]]$; PB là các PM còn lại: $PB = P - PA$
- ✧ Sắp tăng PA theo TG; sắp giảm PB theo TG.
- ✧ Nối PB vào sau PA: $result = PA + PB$

Minh họa

PM	1	2	3	4	5	6
A: $a[i]$	3	2	2	1	1	2
B: $b[i]$	2	2	1	3	2	1
PA: $a[i] \leq b[i]$		2		4	5	
PB: rest	1		3			6
PA $\uparrow a$	4(1) 5(1) 2(2)					
PB $\downarrow b$	1(2) 3(1) 6(1)					
Result	4 5 2 1 3 6					

Giải trình kết quả

phút	1	2	3	4	5	6	7	8	9	10	11	12
A	4	5	2	2	1	1	1	3	3	6	6	
B		4	4	4	5	5	2	2	1	1	3	6

Ca làm việc kết thúc sau 12 phút.

Chú ý

Các chỉ số PM trong đề bài được tính từ 1, nhưng trong chương trình được tính từ 0, nên khi giải trình kết quả bạn nhớ tăng thêm 1.

Program

```
// Johnson Algorithm: Scheduling on two machines
#include<bits/stdc++.h>
using namespace std;
const char * FN = "TEST.INP";
typedef vector<int> VI;
int n; // so PM
VI a, b; // time of two machines
VI pa; // {i: a[i] <= b[i]}
```

```

VI pb; // the rest
inline void Go(const char * msg = " ? ") {
    cout << msg;
    fflush(stdin);
    if (cin.get() == '.') exit(0);
}
void Print(VI v, const char * msg = "") {
    cout << msg;
    for(auto e:v)
        cout << " " << e;
}
void Input() {
    ifstream f(FN);
    if(!f) {
        cout << "\n unable open input file " << FN;
        exit(0);
    }
    f >> n; // so PM
    cout << n << " PM";
    a.assign(6, 0); b.assign(6, 0);
    // time for a
    for(int i = 0; i < n; ++i) f >> a[i];
    // time for b
    for(int i = 0; i < n; ++i) {
        f >> b[i];
        // split
        if (a[i] <= b[i]) pa.push_back(i);
        else pb.push_back(i);
    }
    f.close();
    Print(a, "\n Time on machine A:");
    Print(b, "\n Time on machine B:");
}
inline bool aLess(int x, int y) { return a[x] < a[y]; }
inline bool bGreat(int x, int y) { return b[x] > b[y]; }
void Run() {

```



```

    Input();
    cout << "\n Split:";
    Print(pa, "\n PA:");
    Print(pb, "\n PB:");
    sort(pa.begin(), pa.end(), aLess); // sap tang pa
    sort(pb.begin(), pb.end(), bGreat); // sap giam pb
    Print(pa, "\n inc sort PA:");
    Print(pb, "\n dec sort PB:");
    // pa += pb
    for (auto e:pb)
        pa.push_back(e);
    cout << "\n Result:"; // tang 1 cho index
    for (auto e:pa) cout << " " << e+1;
}
int main() {
    Run();
    cout << endl << " T h e   E n d";
    return 0;
}

```

Output

```

6 PM
Time on machine A: 3 2 2 1 1 2
Time on machine B: 2 2 1 3 2 1
Split:
PA: 1 3 4
PB: 0 2 5
inc sort PA: 3 4 1
dec sort PB: 0 2 5
Result: 4 5 2 1 3 6
T h e   E n d

```

Nearest Points

Cho $n \geq 2$ điểm nguyên khác nhau trên mặt phẳng tọa độ. Tìm một cặp điểm (A, B) , $A \neq B$ có khoảng cách Euclid ngắn nhất. Khoảng cách Euclid giữa 2 điểm $A(ax, ay)$ và $B(bx, by)$ được tính theo công thức:

$$d(A, B) = \sqrt{(ax - bx)^2 + (ay - by)^2}$$

POINT.INP	OUTPUT
7	1.41421
3 7	(2, 6)
2 6	(3, 7)
7 5	
1 3	
7 3	
3 3	
3 1	

Ý nghĩa:

input: Dòng đầu tiên n: số lượng điểm.

Tiếp đến là n dòng, mỗi dòng 2 số nguyên x y biểu thị

tọa độ của điểm.

Output: Dòng đầu tiên số thực d là khoảng cách min tìm được.

Hai dòng tiếp theo: tọa độ của hai điểm A, B có khoảng cách d.

Max(n) = 20000.

Tiếp cận 1: Duyệt toàn bộ

Tính khoảng cách của mọi cặp điểm (i,j), i, j: 1..n giữ lại khoảng cách min.

Gọi mảng chứa n điểm là p, nếu duyệt toàn bộ ta phải tính khoảng cách giữa mỗi điểm i với mỗi điểm j, tức là ta tốn n^2 .

Bạn nên tính khoảng bình phương cách d^2 thay cho d nhằm giảm thời gian lấy căn.

Độ phức tạp

$$O(n^2)$$

Program

```
// Nearest points: Phuong an 1. Vet can
#include<bits/stdc++.h>
using namespace std;
const char * FN = "POINT.INP";
int n; // number of points
inline void Go(const char * msg = " ? ") {
    cout << msg;
    fflush(stdin);
    if (cin.get() == '.') exit(0);
}
class Point {
public:
    int X, Y;
    Point() { X = 0; Y = 0; }
    Point(int x, int y) { X = x; Y = y; }
    void Set(int x, int y) { X = x; Y = y; }
    // bình phương Khoảng cách den goc toa do
    int D2() { return X*X+Y*Y; }
```

```

        void Print(const char * msg = "") {
            cout << msg << "(" << X << "," << Y << ")";
        }
    };

typedef vector<Point> VP;
VP p;
void Print(VP v, const char * msg = "") {
    cout << msg;
    for(auto p:v) p.Print(" ");
}

void Input() {
    ifstream f(FN);
    if(!f) {
        cout << "\n unable open input file " << FN;
        exit(0);
    }
    f >> n; // number of points
    cout << n << " points";
    Point xy;
    p.assign(n, xy);
    int x, y;
    for(int i = 0; i < n; ++i) { f >> x >> y; p[i].Set(x,y); }
    f.close();
}

Point Minus(Point a, Point b) { return Point(a.X-b.X, a.Y-b.Y); }
int Distance2(Point a, Point b) { return Minus(a, b).D2(); }
void Run() {
    Input();
    Print(p, "\n p:");
    int dmin = Distance2(p[0], p[1]);
    int imin = 0, jmin = 1;
    for(int i = 1; i < p.size(); ++i) {
        for(int j = 0; j < i; ++j) {
            int d = Distance2(p[i], p[j]);
            if (d < dmin) { dmin = d; imin = i; jmin = j; }
        } // for j
    } // for i
    cout << "\n Result: \n " << dmin; // 1.41421
    p[imin].Print("\n "); // (2, 6)
}

```

```

p[jmin].Print("\n "); // (3, 7)
}
int main() {
    Run();
    cout << endl << " T h e   E n d";
    return 0;
}

```

Output

```

7 points
p: (3,7) (2,6) (7,5) (1,3) (7,3) (3,3) (3,1)
Result:
2
(3,7)
(2,6)
T h e   E n d

```

Tiếp cận 2: Chia để trị

Giả sử dãy điểm $p[1..n]$ đã được sắp tăng theo toạ độ x . Ta lấy điểm giữa $p[m]$ của dãy và xét 2 nửa, trái $LS = p[0..m]$ và phải $RS = p[m+1..n-1]$ của dãy p . Giả sử, ta đã xác định được giá trị L và R là khoảng cách min của các cặp điểm trong nửa trái LS và nửa phải RS tương ứng, và ta đã lấy $dmin = \min(L, R)$. Ta thấy $dmin$ chưa phải là kết quả trên toàn p . Lý do là ta chưa xét tương quan giữa các điểm của nửa trái và nửa phải. Việc xét này hóa ra lại rất đơn giản.

Tạm ký hiệu $d(u,v)$ là khoảng cách giữa 2 điểm u và v , ta để ý rằng, nếu các điểm e của nửa trái hoặc nửa phải nằm quá xa điểm $p[m]$, tức là $d(e, p[m]) \geq dmin$ thì ta bỏ qua. Vậy ta chỉ cần xét các điểm lân cận trái a và lân cận phải b của $p[m]$. Lúc này ta tính khoảng cách giữa mỗi điểm lân cận trái a và mỗi điểm lân cận phải b và cập nhật $dmin$. Ta có thuật toán sau:

1. Sắp tăng dãy p theo x , Độ phức tạp $O(n \log(n))$
2. Thực hiện thủ tục tựa merge sort trên mỗi khoảng $d..c$:
 - 2.1. Chia đôi đoạn $d..c$, điểm giữa $m = (d+c)/2$
 - 2.2. Nạp các điểm i bên trái gần m tức là các điểm thỏa hệ thức

$$d(p[m], p[i]) < dmin \text{ vào dãy con left (*)}$$
 - 2.3. Duyệt nửa phải, xét các điểm j bên phải gần m ,

đổi sánh chúng với mỗi điểm i trong tập left.

Độ phức tạp $O(\log(n))$

Bạn có thể tạm thay hàm tính khoảng cách bằng hàm bình phương khoảng cách Distance2(a,b), khi giải trình kết quả bạn sẽ lấy căn bậc hai sau.

Chương trình dưới đây cài đặt thêm hàm Autorun nhằm sinh tự động dữ liệu như sau: Hàm sinh tối đa (20000) điểm có tọa độ là các số nguyên chẵn có hoành độ (x) cách nhau 2 đơn vị và phân bố đều trong một hình vuông. Ngoài ra hàm sinh thêm một điểm lẻ p[i]. Kết quả dự đoán phải là dmin = 1.0.

Program

```
// Nearest points: Phuong an 2: Chia de tri
#include<bits/stdc++.h>
using namespace std;
const char * FN = "POINT.INP";
typedef vector<int> VI;
int n; // number of points
int dmin, imin, jmin; // min value and indices
inline void Go(const char * msg = " ? ") {
    cout << msg;  fflush(stdin);
    if (cin.get() == '.') exit(0);
}

class Point { . . . };
typedef vector<Point> VP;
VP p;
void Print(VP v, const char * msg = "") {
    cout << msg; for(auto p:v) p.Print(" ");
}
void Input() {
    ifstream f(FN);
    if(!f) {
        cout << "\n unable open input file " << FN;
        exit(0);
    }
    f >> n;
    cout << n << " points";
    Point xy;
    p.assign(n, xy);
    int x, y;
    for(int i = 0; i < n; ++i) {
```

```

        f >> x>> y;
        p[i].Set(x,y);
    }
    f.close();
}

Point Minus(Point a, Point b) { return Point(a.X-b.X, a.Y-b.Y); }
int Distance2(Point a, Point b) { return Minus(a, b).D2(); }
bool Less(Point a, Point b) { return a.X < b.X; }
// chia tap diem p[d..c] thanh 2 nua,
// nua trai p[d..m] va nua phai a[m+1..c]
void Find(int d, int c) {
    VI LS, RS;
    if (d >= c) return;
    int m = (d + c) / 2;
    Find(d,m); Find(m+1,c);
    // ghi nhan cac phan tu gan m o nua trai p[d..m] vao left id
    for(int i = m; i >= d; --i) {
        if (Distance2(p[i],p[m]) < dmin)
            LS.push_back(i);
    }
    // ghi nhan cac phan tu gan m o nua phai p[m+1..c] vao right id
    for(int j = m+1; j <= c; ++j) {
        if (Distance2(p[j],p[m]) < dmin) RS.push_back(j);
    }
    // so sanh moi diem gan m cua nua phai
    // voi moi diem trong nua trai
    for(int j = 0; j < RS.size(); ++j) {
        int jj = RS[j];
        for (int i = 0; i < LS.size(); ++i) {
            int ii = LS[i];
            int dt = Distance2(p[ii], p[jj]);
            if (dt < dmin) {dmin = dt; imin = ii; jmin = jj; }
        } // for i
    } // for j
}

void GetMin() {
    dmin = Distance2(p[0], p[1]); // Init dmin

```

224 CHUYÊN TIN 11: CÁC CHUYÊN ĐỀ CHỌN LỌC VỚI NGÔN NGỮ C++

```
    imin = 0; jmin = 1; // Init imin, jmin
    Find(0, p.size()-1);
    cout << "\n Result: " << dmin << " -> " << sqrt(dmin);
    p[imin].Print("\n "); p[jmin].Print("\n ");
}

void Run() {
    Input();
    Print(p, "\n p:");
    sort(p.begin(), p.end(), Less);
    Print(p, "\n Sorted p:");
    GetMin();
}

void Autorun(int n) {
    time_t t = time(NULL);
    cout << "\n Autorun: n = " << n;
    // sinh data
    p.clear();
    int i, v;
    for (i = 1; i < n; ++i) v = i+i; p.push_back(Point(v,v));
    i = n; // 100; v = p[i][X]
    p.push_back(Point(v,v+1));
    GetMin();
    cout << endl << " Time: " << difftime(time(NULL), t);// sec.
}

int main() {
    Run();
    Autorun(20000);
    cout << endl << " T h e   E n d";
    return 0;
}
```

Output

```
7 points
p: (3, 7) (2, 6) (7, 5) (1, 3) (7, 3) (3, 3) (3, 1)
Sorted p: (1, 3) (2, 6) (3, 7) (3, 3) (3, 1) (7, 5) (7, 3)
Result: 2 -> 1.41421
(2, 6)
(3, 7)
Autorun: n = 20000
```

Result: 1 -> 1

(39998, 39998)

(39998, 39999)

Time: 0

T h e E n d

Longest Common Substring

Cho hai string x và y dài tối đa 2000. *Xâu con của x* là *xâu z nhận được từ x sau khi xóa một số ký tự của x* . Giả thiết rằng sau khi xóa một số ký tự của x thì các ký tự còn lại của x sẽ xích lại sát nhau. Hãy xác định chiều dài của *xâu con chung dài nhất của x và y* .

Ví dụ

$x = \mathbf{a1b2c3d4}$, $y = \mathbf{eafbghcidjm}$, $z = \mathbf{abcd}$ là *xâu con chung dài nhất của x và y* , $\text{len}(z) = 4$.

Kết quả: 4

Algorithm

Quy hoạch động

Cho *xâu $x = (x_0, x_1, \dots, x_{n-1})$* . Ta ký hiệu $x(i)$ là *tiền tố* gồm $i+1$ phần tử đầu tiên của x , $x(i) = (x_0, x_1, \dots, x_i)$. Xét hàm 2 biến $S(i, j)$ là *đáp số* khi giải bài toán với 2 *tiền tố $x(i)$ và $y(j)$* . Ta có,

Phương trình Bellman (Hệ thức quy hoạch động):

☞ if $i < 0$ or $j < 0$ then $S(i, j) = 0$: một trong hai *xâu* là *rỗng* thì *xâu con chung* là *rỗng*

☞ else // $i \geq 0$ và $j \geq 0$

➤ if $x[i] = y[j]$ then $S(i, j) = S(i-1, j-1) + 1$

➤ else // $x[i] \neq y[j]$: $S(i, j) = \max(S(i-1, j), S(i, j-1))$

x :----- $x[i] = a$

y :----- $y[j] = a$

x :----- $x[i] = a$

y :----- $y[j] = b$

Version 1: Recursive

```
// LCS: Longest Common Substring
```

```
// Version 1: Recursive
```

```
#include<bits/stdc++.h>
```

```
using namespace std;
```

```
string x, y;
```


226 CHUYÊN TIN 11: CÁC CHUYÊN ĐỀ CHỌN LỌC VỚI NGÔN NGỮ C++

```
// Recursive
int S(int i, int j) {
    if (i < 0 or j < 0) return 0;
    return (x[i] == y[j]) ? S(i-1,j-1)+1
        : max(S(i-1, j),S(i, j-1));
}

void Run(string xx, string yy) {
    x = xx; y = yy;
    cout << "\n Result: " << S(x.length()-1, y.length()-1);
}

int main() {
    Run("a1b2c3d4", "eafbghcidjm"); // 4
    cout << endl << " T h e   E n d";
    return 0;
}
```

Output

```
Result: 4
T h e   E n d
```

Độ phức tạp

Rất lớn vì quá nhiều lời gọi hàm trùng lặp.

Cách tạo test

- ✂ Bước 1. Sinh string z dài tùy ý
- ✂ Bước 2. Xen vào z một số ký tự để nhận được string x
- ✂ Bước 3. Xen vào z một số ký tự để nhận được string y

Version 2: Using an 2D array s

Thay vì gọi các hàm đệ quy $S[i][j]$ ta tổ chức điền mảng s theo trật tự từ dòng 0 đến dòng $\text{len}(x) - 1$. Tại mỗi dòng, ta điền từ trái qua phải.

Độ phức tạp

$O(\text{Len}(x) * \text{len}(y))$

Program

```
//LCS: Longest Common Substring
// Version 2: Using 2D array
#include<bits/stdc++.h>
using namespace std;
string x, y;
```

```

int S2() { // using 2D array
    cout << "\n Version 2: using an array 2D";
    int n = x.length();
    int m = y.length();
    if (n == 0 || m == 0) return 0;
    int s[n][m];
    memset(s, 0, n*m*sizeof(int));
    // Init row s[0][j]
    for (int j = 0; j < m; ++j)
        if (x[0] == y[j]) s[0][j] = 1;
    // set values
    for (int i = 1; i < n; ++i) {
        if (x[i] == y[0]) s[i][0] = 1;
        for (int j = 1; j < m; ++j) {
            s[i][j] = (x[i] == y[j]) ? s[i-1][j-1] + 1
                : max(s[i][j-1], s[i-1][j]);
        } // for j
    } // for i
    return s[n-1][m-1];
}

void Run(string xx, string yy) {
    x = xx; y = yy;
    cout << "\n Result: " << S2();
}

int main() {
    Run("a1b2c3d4", "eafbgchcidjm"); // 4
    cout << endl << " T h e   E n d";
    return 0;
}

```

Output

```

Result:
Version 2: using an array 2D4
T h e   E n d

```

Version 3: Using two 1D arrays a , b

Dùng hai mảng một chiều a: dòng trên của ma trận s, b: dòng dưới của s.

Sau mỗi vòng lặp i ta hoán vị a và b như hai con trỏ chứ không dữ liệu.

Độ phức tạp

$$O(\text{Len}(x) * \text{len}(y))$$

Program

```
//LCS: Longest Common Substring
// Version 3: Using two 1D arrays a, b
#include<bits/stdc++.h>
using namespace std;
string x, y;
int S3() { // using 1D array
    cout << "\n Version 3: using an array 1D";
    int n = x.length(), m = y.length();
    if (n == 0 || m == 0) return 0;
    int *a, *b, *c;
    a = new int[m]; b = new int[m];
    memset(a, 0, m*sizeof(int));
    memset(b, 0, m*sizeof(int));
    // Init a: a[0] b[j]
    for (int j = 0; j < m; ++j)
        if (x[0] == y[j]) a[j] = 1;
    for (int i = 1; i < n; ++i) {
        b[0] = (x[i] == y[0]) ? 1 : 0;
        for (int j = 1; j < m; ++j) {
            b[j] = (x[i] == y[j]) ? a[j-1] + 1
                : max(b[j-1], a[j]);
        } // for j
        c = a; a = b; b = c;
    } // for i
    return a[m-1];
}

void Run(string xx, string yy) {
    x = xx; y = yy;
    cout << "\n Result: " << S3();
}

int main() {
    Run("a1b2c3d4", "eafbghcidjm"); // 4
    cout << endl << " T h e   E n d";
    return 0;
}
```

Output

Result:

Version 3: using an array 1D4

T h e E n d

Longest Monotone Segment

Cho dãy n số nguyên a . Một đoạn (i, j) , $i \leq j$ trong a bao gồm các phần tử từ $a[i]$ đến $a[j]$ kể cả hai phần tử $a[i]$ và $a[j]$. Đoạn (i, j) được gọi là đơn điệu tăng nếu $a[i] \leq a[i+1] \leq \dots \leq a[j-1] \leq a[j]$, đơn điệu giảm nếu $a[i] \geq a[i+1] \geq \dots \geq a[j-1] \geq a[j]$. Đoạn (i, j) được gọi là đơn điệu nếu (i, j) là đơn điệu tăng hoặc giảm. Hãy tìm đoạn đơn điệu dài nhất trong a ?

Giới hạn

$$2 \leq n \leq 2000$$

Chỉ số tính từ 0

Ví dụ

MONO.INP	OUTPUT	Giải thích
10	2 6	7 5 5 4 2
2 2 7 5 5		đoạn đơn điệu dài nhất
4 2 9 9 3		từ $a[2]$ đến $a[6]$

Algorithm

E. Dijkstra đề xuất thuật toán $O(n)$ duyệt một lần xác định được đoạn đơn điệu dài nhất trong dãy a như sau:

Dùng hai biến t đếm các trường hợp đi lên: $a[i-1] < a[i]$, g đếm các trường hợp đi xuống $a[i-1] > a[i]$. Dễ thấy là khi $a[i-1] == a[i]$ ta vẫn phải tăng t và g (theo định nghĩa của đồng biến và nghịch biến). Sau mỗi bước i ta tính max của ba giá trị $\maxlen$, t và g và cập nhật $\maxlen$ và vị trí đạt max là imax . Bạn nhớ khởi trị $t = g = 1$ vì dãy có 1 phần tử là đơn điệu với chiều dài 1.

Cách sinh test

Để test thuật toán bạn sinh dữ liệu theo mẫu là một string form chứa các ký tự 'b', 't' và 'g' cho biết là cần sinh ra các số bằng, lớn hơn hay nhỏ thua số sát trước. Ví dụ:

form = "bbtggg" sẽ sinh ra dãy 6 số

$a = [0, 0, 1, 0, -1, -2]$

Program

```
// LMS: Longest Monotone Segment
#include<bits/stdc++.h>
```

230 CHUYÊN TIN 11: CÁC CHUYÊN ĐỀ CHỌN LỌC VỚI NGÔN NGỮ C++

```

using namespace std;
const char * FN = "MONO.INP";
typedef vector<int> VI;
VI a;
inline void Go(const char * msg = " ? ") {
    cout << msg;  fflush(stdin);
    if (cin.get() == '.') exit(0);
}
void Print(VI v, const char * msg = "") {
    cout << msg;
    for(auto e:v)  cout << " " << e;
}
void Input() {
    ifstream f(FN);
    int n;
    f >> n;
    a.clear(); a.assign(n, 0);
    for (int i = 0; i < n; ++i) f >> a[i];
    f.close();
}
void LMS() { // Longest Monotone Segment
    int t = 1, g = 1;
    int m, maxlen = 0;
    int maxi; // diem ket
    for (int i = 1; i < a.size(); ++i) {
        if (a[i] == a[i-1]) { ++t; ++g; }
        else {
            if (a[i-1] < a[i]) { ++t; g = 1; }
            else { t = 1; ++g; }
        }
        m = max(t, g);
        if (m > maxlen) { maxlen = m; maxi = i; }
    } // for i
    cout << "\n maxlen = " << maxlen;
    cout << "\n maxi = " << maxi;
    cout << "\n Result: " << " " << maxi-maxlen+1 << " " << maxi;
}

```

```

void AutoTest() {
    string form;
    while(true) {
        cout << "\n\n Auto test ";
        cout << "\n Input form g t b: ";
        fflush(stdin);
        cin >> form;
        int n = form.length();
        cout << "\n len of form = " << n;
        a.clear(); a.assign(n, 0);
        a[0] = 0;
        for (int j = 1; j < n; ++j) {
            switch(form[j]) {
                case 'b': a[j] = a[j-1]; break;
                case 't': a[j] = a[j-1] + 1; break;
                case 'g': a[j] = a[j-1] - 1; break;
            }
        }
        Print(a, "\n Init a: ");
        LMS();
        Go("\n Press dot key [.] to stop: ");
    }
}

void Run() {
    Input(); Print(a); LMS();
}

int main() {
    Run();
    AutoTest();
    cout << endl << " T h e   E n d";
    return 0;
}

```

Output

```

2 2 7 5 5 4 2 9 9 3
maxlen = 5
maxi = 6
Result: 2 6
Auto test

```

232 CHUYÊN TIN 11: CÁC CHUYÊN ĐỀ CHỌN LỌC VỚI NGÔN NGỮ C++

```
Input form g t b: bbbbbbgggggggtttttttttttttttttttttt
len of form = 35
Init a:  0 0 0 0 0 0 -1 -2 -3 -4 -5 -6 -7 -6 -5 -4 -3 -2 -1 0 1 2 3 4 5 6 7
8 9 10 11 12 13 14 15
maxlen = 23
maxi = 34
Result:  12 34
Press dot key [.] to stop:
Auto test
Input form g t b: tttttttttbbbbbbbbbbggg
len of form = 22
Init a:  0 1 2 3 4 5 6 7 8 8 8 8 8 8 8 8 8 8 7 6 5
maxlen = 19
maxi = 18
Result:  0 18
Press dot key [.] to stop: .
```

Longest Repeated Sliding Segments

Bentley: Những viên ngọc lập trình

Cho dãy n phần tử $x = (x_0, x_1, \dots, x_{n-1})$ và hai chỉ số i, j , $0 \leq i < j < n$. Ký hiệu $x[i:i] = (x_0, x_1, \dots, x_{i-1})$ là tiền tố của x , $x[i:] = (x_i, x_{i+1}, \dots, x_{n-1})$ là hậu tố của x , $x[i:j] = (x_i, x_{i+1}, \dots, x_{j-1})$ là một đoạn của x .

Cho string x dài n ký tự. Hãy xác định hai đoạn $u = v$ dài nhất trong x , u bắt đầu xuất hiện tại vị trí i và v xuất hiện tại vị trí j , $i < j$.

Ví dụ $x = "xabababayyy"$ cho ta $u = v = ababa$, u bắt đầu xuất hiện tại $i = 1$, v bắt đầu xuất hiện tại $j = 3$. Ta nói đoạn $u = ababa$ trượt từ vị trí $i = 1$ đến vị trí $j = 3$.

Input: string x

Output:

☞ $i\ j$: chỉ số đầu u và v

☞ m : chiều dài max của u (và v)

Giới hạn

☞ $2 \leq n \leq 2000$

☞ Các chỉ số tính từ 0

Ví dụ trên cho kết quả

1 3

5

Algorithm 1

Xét hàm 2 biến $S(i,j)$ cho ra chiều dài max của hai khúc đầu khi so sánh hai hậu tố $x[i:]$ và $x[j:]$, $i < j$. Ví dụ, $x = \text{"xabababayyy"}$ cho ta:

```
S(0,1) = 0, vì
x[0:] = "xabababayyy"
x[1:] = "abababayyy"
```

```
S(1, 3) = 5. vì
x[1:] = "abababayyy"
x[5:] = "abayyy"
```

Ta có:

```
// khúc đầu trùng nhau dài nhất của x[i:] và x[j:], i < j
int S(int i, int j) {
    int k = i;
    for (;j < x.length(); ++j) {
        if (x[j] != x[k]) break;
        ++k; // x[j] == x[k]
    }
    return k-i;
}
```

Hàm $S(i,j)$ có độ phức tạp $O(n)$.

Việc còn lại là thực hiện duyệt tìm các $S(i,j)$ với mỗi $i = 0:\text{len}(s)-k_{\max}$ và mỗi $j = i+1:\text{len}(x)$, trong đó $k_{\max}$ là chiều dài lớn nhất tìm được sau mỗi bước.

Độ phức tạp của phương án 1

$O(n^3)$. Gọi n là chiều dài của input. Việc thực hiện các $S(i,j)$ đòi hỏi n^2 lần lặp, mỗi lần lặp cần n phép duyệt.

Program

```
// LRSS: Longest Repeated Sliding Segments
// Version 1.  $O(n^3)$ 
#include<bits/stdc++.h>
using namespace std;
string x;
// khúc đầu trùng nhau dài nhất của x[i:] và x[j:], i < j
int S(int i, int j) {
    int k = i;
    for (;j < x.length(); ++j) {
```


234 CHUYÊN TIN 11: CÁC CHUYÊN ĐỀ CHỌN LỌC VỚI NGÔN NGỮ C++

```
        if (x[j] != x[k]) break;
        ++k;
    }
    return k-i;
}

void Run1(string xx) { // vet can
    x = xx;
    cout << "\n Input: " << x;
    int n = x.length();
    int kmax = 0, imax = 0, jmax = 0;
    for(int i = 0; i < n-kmax; ++i) {
        for(int j = i+1; j < n; ++j) {
            int k = S(i,j);
            if (k > kmax) { kmax = k; imax = i; jmax = j; }
        }
    }
    cout << "\n " << imax << " " << jmax << " " << kmax;
}

int main() {
    Run1("a*****bc"); // 1 2 7
    Run1("x123456aba12345cd123456aba1"); // 1 17 10
    Run1("x12345aba12345"); // 1 9 5
    Run1("xabababayyy"); // 1 3 5
    cout << endl << " T h e   E n d";
    return 0;
}
```

Output

```
Input: a*****bc
1 2 7
Input: x123456aba12345cd123456aba1
1 17 10
Input: x12345aba12345
1 9 5
Input: xabababayyy
1 3 5
T h e   E n d
```

Algorithm 2 (Bentley)

- ☞ Sắp tăng theo chỉ dẫn các hậu tố của s theo trật tự từ điển. Gọi dãy được sắp theo chỉ dẫn này là $\text{id}[1..n]$, $n = \text{len}(s)$.
- ☞ Duyệt dãy được sắp trong id, với mỗi cặp hậu tố đứng kề nhau $x[\text{id}[i-1]:]$ và $x[\text{id}[i]:]$, $i = 1..n-1$ ta gọi hàm $\text{Compare}(\text{id}[i-1], \text{id}[i])$ để tính chiều dài của khúc đầu dài nhất của chúng.

Độ phức tạp của phương án 2

Về miền nhớ: $O(n^2)$. Ta phải lưu trữ các hậu tố của xâu x chiều dài n, tức là phải thêm:

$$n + (n-1) + \dots + 1 = (n+1)*n/2 \text{ ký tự}$$

Về tốc độ tính: $O(n^2 \log(n))$. Sắp tăng các hậu tố cần $\log(n)$ lần lặp, mỗi lần lặp cần n phép so sánh hai strings, mỗi phép so sánh hai strings cần duyệt n cặp ký tự.

Ví dụ: $x = \text{"xabababayyy"}$

Sau khi sắp tăng các hậu tố $x[i:]$, $i = 0..n-1$ theo chỉ số ta thu được:

Trước khi sắp	Sau khi sắp	max len dòng trên và dưới
$x[0:] = \text{"xabababayyy"}$	$x[1:] = \text{"abababayyy"}$	
$x[1:] = \text{"abababayyy"}$	$x[3:] = \text{"ababayyy"}$	5
$x[2:] = \text{"bababayyy"}$	$x[5:] = \text{"abayyy"}$	3
$x[3:] = \text{"ababayyy"}$	$x[7:] = \text{"ayyy"}$	1
$x[4:] = \text{"babayyy"}$	$x[2:] = \text{"bababayyy"}$	0
$x[5:] = \text{"abayyy"}$	$x[4:] = \text{"babayyy"}$	4
$x[6:] = \text{"bayyy"}$	$x[6:] = \text{"bayyy"}$	2
$x[7:] = \text{"ayyy"}$	$x[0:] = \text{"xabababayyy"}$	0
$x[8:] = \text{"yyy"}$	$x[10:] = \text{"y"}$	0
$x[9:] = \text{"yy"}$	$x[9:] = \text{"yy"}$	1
$x[10:] = \text{"y"}$	$x[8:] = \text{"yyy"}$	2

Program

```
// LRSS: Longest Repeated Sliding Segments
// Version 2. sort (Bentley)
#include<bits/stdc++.h>
using namespace std;
typedef vector<int> VI;
typedef vector<string> VS;
VI id;
string x;
VS sub;
```

```

// Khuc dau trung nhau dau nhat
// cua sub[id[i]] va sub[id[i-1]]
int Compare(int i) {
    string a = sub[id[i]];
    string b = sub[id[i-1]];
    int minlen = min(a.length(), b.length());
    for(int j = 0; j < minlen; ++j)
        if (a[j] != b[j]) return j;
    return minlen;
}

bool Less(int i, int j) {
    return sub[i] < sub[j];
}

// duyet sub sap theo id
// tim max len 2 day lien ke
void Result() {
    int maxlen = 0, maxi = 0, maxj = 0;
    for(int i = 1; i < id.size(); ++i) {
        int len = Compare(i);
        if (len > maxlen) {
            maxlen = len;
            maxi = id[i];
            maxj = id[i-1];
        }
    }
    cout << "\n Result: " << " " << maxi << " " << maxj
        << " " << maxlen << endl;
}

void Run2(string xx) {
    string x = xx;
    int n = x.length();
    cout << "\n Input: " << x << " len = " << n;
    id.assign(n, 0);
    sub.clear();
    for(int i = 0; i < n; ++i) {
        id[i] = i;
        sub.push_back(x.substr(i));
    }
}

```

```

sort(id.begin(), id.end(), Less); // sort theo id
cout << "\n sorted: ";
for(auto i:id) cout << "\n " << sub[i];
Result();
}

int main() {
    Run2("xabababayyy"); // 3 1 5
    Run2("a*****bc"); // 2 1 7
    Run2("x123456aba12345cd123456aba1"); // 1 17 10
    Run2("x12345aba12345"); // 1 9 5
    cout << endl << " T h e   E n d";
    return 0;
}

```

Output

Input: xabababayyy len = 11

sorted:

abababayyy

ababayyy

abayyy

ayyy

bababayyy

babayyy

bayyy

xabababayyy

y

yy

yyy

Result: 3 1 5

Input: a*****bc len = 11

sorted:

*****bc

*****bc

*****bc

*****bc

****bc

***bc

238 CHUYÊN TIN 11: CÁC CHUYÊN ĐỀ CHỌN LỌC VỚI NGÔN NGỮ C++

****bc**

***bc**

a***bc**

bc

c

Result: 2 1 7

Input: x123456aba12345cd123456aba1 len = 27

sorted:

1

123456aba1

123456aba12345cd123456aba1

12345cd123456aba1

23456aba1

23456aba12345cd123456aba1

2345cd123456aba1

3456aba1

3456aba12345cd123456aba1

345cd123456aba1

456aba1

456aba12345cd123456aba1

45cd123456aba1

56aba1

56aba12345cd123456aba1

5cd123456aba1

6aba1

6aba12345cd123456aba1

a1

a12345cd123456aba1

aba1

aba12345cd123456aba1

ba1

ba12345cd123456aba1

cd123456aba1

d123456aba1

x123456aba12345cd123456aba1

Result: 1 17 10

Input: x12345aba12345 len = 14

sorted:

12345

12345aba12345

2345

2345aba12345

345

345aba12345

45

45aba12345

5

5aba12345

a12345

aba12345

ba12345

x12345aba12345

Result: 1 9 5

T h e E n d

Dictionary

Olympic Moscow

Các từ trong bài được hiểu là một dãy liên tiếp các chữ ký tự. Một file văn bản chứa một từ điển D gồm tối đa $n = 100$ từ khác nhau đôi một. Mỗi từ dài không quá 50 ký tự và được viết trên một dòng. Dòng cuối của file là từ s dài không quá 200 ký tự. Hãy cho biết cần xóa đi khỏi s tối thiểu bao nhiêu chữ cái để phần còn lại tạo thành dãy liên tiếp các từ trong từ điển D , mỗi từ có thể xuất hiện nhiều lần.

Ví dụ:

DIC.INP	DIC.OUT
6	5
abba	
not	
is	
astra	
saint	
panama	
saintpavnamtranaisnotsaintabba	

Sau khi xóa 5 chữ cái (gạch dưới)

saintpavnamtranaisnotsaintabba

ta thu được dãy ghép của các từ 4, 5, 2,
1, 4, 0

saintpanamaisnotsaintabba

*Algorithm**Dynamic programming*

Gọi d là tập n từ trong từ điển, s là từ cần xử lý. Gọi $\text{GetMax}(i)$ là hàm giải bài toán với tiền tố $s[:i]$. Hàm cho ra chiều dài max len của dãy liên tiếp các từ trong từ điển d tạo thành một dãy con trong $s[:i]$. Với mỗi từ w dài m ký tự trong từ điển d , ta xét hàm $\text{Arrange}(w, i)$ có chức năng rải ngược w vào tiền tố $s[:i]$ như sau. Hàm cho ra chỉ số v thỏa hai điều kiện sau:

✓ w là xâu con của $s[v:i]$, nghĩa là nếu xóa đi một số ký tự khỏi $s[:i]$ ta sẽ thu được từ w ,

✓ $s[v] = w[0]$, $s[i] = w[m-1]$.

Đặt $p[i] = \text{GetMax}[i]$

Ta thu được kết quả $p[n-1]$ sau n lần gọi hàm $\text{GetMax}(i)$, $i = 0..n-1$.

Khi đó số ký tự cần xóa khỏi s sẽ là $n - p[n-1]$.

```
// Nhung tu w vao tien to s:i
// return v: w is substr of s[v..i]
// s[v] = w[0]
int Arrange(const string &w, int i) {
    int j = w.length()-1;
    if (i < j) return -1;
    for (; i >= 0; --i)
        if (w[j] == s[i]) {
            --j;
            if (j < 0) return i;
        }
    return -1;
}
```

Nếu w được nhúng trong $s[:i]$ thì số ký tự được giữ lại trong $s[:i]$ sẽ là $\text{len}(w)$. Ta cần chọn w sao cho giá trị này đạt max.

$$d(i) = \max \{ d(v-1) + \text{len}(w) \mid w \in d, v = \text{Arrange}(w, i) \}$$

Khi w không thể nhúng được trong $s[..i]$ ta đặt $v = \text{Arrange}(w, i) = -1$.

Program

```
// Dictionary
#include<bits/stdc++.h>
using namespace std;
const char * FN = "DIC.INP";
```

```

typedef vector<string> VS;
VS d; // dictionary
typedef vector<int> VI;
VI p; // chiều dài max sau khi xóa các ký tự ngoại w
int n; // số từ trong từ điển
string s; // từ cần xử lý
int lens; // chiều dài s
void Print(VS v, const char * msg = "") {
    cout << msg;
    int c = -1;
    for(auto s:v) { ++c; cout << c << ". " << s << endl; }
}
void Input() {
    ifstream f(FN);
    if(!f) {
        cout << "\n unable open input file " << FN;
        exit(0);
    }
    d.clear();
    f >> n;
    cout << "\n n = " << n;
    string x;
    for(int i = 0; i < n; ++i) { f >> x; d.push_back(x); }
    f >> s; // the last word
    f.close();
    Print(d, "\n Dictionary: \n");
    lens = s.length();
    cout << "\n Input s = " << s << "    lens = " << lens;
}
// Nhung từ w vào tiền tố s[:i]
int Arrange(const string &w, int i) {
    int j = w.length()-1;
    if (i < j) return -1;
    for (; i >= 0; --i)
        if (w[j] == s[i]) {
            --j;
            if (j < 0) return i;
        }
}

```



```

    return -1;
}
// xet hau to s[..i]
// duyet cac tu w in d rai tu w
// de dat min
void GetMax(int i) {
    int k, v;
    p[i] = (i == 0) ? 0 : p[i-1];
    for (auto w:d) { // duyet tu w = d[j]
        if ((v = Arrange(w,i)) >= 0) { // veu dat duoc tu w
            k = (v == 0) ? w.length() : p[v-1] + w.length();
            p[i] = max(p[i], k);
        }
    }
}
void Run() {
    Input(); // n, d, s, lens
    p.clear();
    p.assign(lens, 0);
    for (int i = 0; i < lens; ++i) GetMax(i);
    cout << "\n Total deleted smbols: " << lens - p[lens-1];
}
int main() {
    Run();
    cout << endl << " T h e   E n d";
    return 0;
}

```

Output

```

n = 6
Dictionary:
0. abba
1. not
2. is
3. astra
4. saint
5. panama

```

```
Input s = saintpavnamtranaisnotsaintabba    lens = 30
```

```
Total deleted smbois: 5
```

```
T h e    E n d
```

Độ phức tạp

$O(kmn)$, trong đó $k = \text{len}(s)$, n là số từ trong từ điển d , m là chiều dài của từ dài nhất trong d .

Cải tiến

Bạn có thể cải tiến chương trình như sau. Khi đọc dữ liệu và tổ chức từ điển d bạn có thể loại trước khỏi d những từ w nào mà ký tự đầu tiên $w[0]$ hoặc ký tự cuối cùng $w[m]$ không xuất hiện trong s vì khi đó chắc chắn là hàm Arrange sẽ cho giá trị -1 .

Cách sinh Test

- ✂ Viết dãy từ chuẩn x . Ví dụ $x = \text{tomjerry}$
- ✂ Cắt x thành các từ có nghĩa đưa vào từ điển d : tom jerry
- ✂ Thêm vào d một số từ: key, ha, wom
- ✂ Chèn thêm một số chữ cái vào x để làm từ cần xử lý: htaomjkeyrry
- ✂ Ghi vào file DIC.INP

DIC.INP
5
jerry
key
ha
wom
tom
htaomjkeyrry

Ví dụ này đòi hỏi xóa 4 ký tự để thu được tomjerry.

Chương 5

THIẾT KẾ VÀ PHÁT TRIỂN MỘT SỐ SẢN PHẨM PHẦN MỀM ỨNG DỤNG

Sudoku

Sudoku là trò chơi Nhật Bản. Đó là một bảng 9×9 ô trong đó có một số ô đã chứa sẵn một vài số trong khoảng 1 đến 9. Người chơi cần điền nốt các ô còn lại (hiện chứa số 0). Kết quả cuối cùng phải là bảng số đáp ứng được các yêu cầu sau đây:

Mọi dòng, mọi cột và mọi khối con 3×3 phải chứa đầy đủ các số từ 1 đến 9.

Trong ví dụ dưới đây, các số điền sẵn được tô đậm, bạn chỉ được phép thay đổi trị của các ô 0.

4	0	0	0	0	0	6	0	0
0	9	0	0	8	0	0	5	0
5	0	0	0	0	9	2	0	0
6	0	0	0	0	0	1	0	0
0	2	0	0	7	0	0	3	0
0	0	4	0	0	0	0	0	9
0	0	1	3	0	0	0	0	8
0	3	0	0	2	0	0	4	0
0	0	9	0	0	0	0	0	6

Đề bài

4	1	8	2	5	3	6	9	7
3	9	2	6	8	7	4	5	1
5	6	7	4	1	9	2	8	3
6	8	3	5	9	4	1	7	2
9	2	5	1	7	6	8	3	4
1	7	4	8	3	2	5	6	9
7	4	1	3	6	5	9	2	8
8	3	6	9	2	1	7	4	5
2	5	9	7	4	8	3	1	6

Đáp án

SUDOKU.INP								
4	0	0	0	0	0	6	0	0
0	9	0	0	8	0	0	5	0
5	0	0	0	0	9	2	0	0
6	0	0	0	0	0	1	0	0
0	2	0	0	7	0	0	3	0
0	0	4	0	0	0	0	0	9
0	0	1	3	0	0	0	0	8
0	3	0	0	2	0	0	4	0
0	0	9	0	0	0	0	0	6

*Algorithm**Back Tracking*

Sau khi đọc dữ liệu vào mảng s , ta gọi hàm Find để chỉnh lại giá trị cần điền cho ô (i,j) lúc đầu là trống. Nếu giá trị hiện hành của ô trống này là v thì ta duyệt từ $v+1$ đến 9 để tìm giá trị đầu tiên c thỏa điều kiện:

- ✧ Dòng i không chứa c ;
- ✧ Cột j không chứa c ;
- ✧ Khối 3×3 chứa ô (i,j) không chứa c .
- Nếu tìm được giá trị c như trên, ta điền c vào ô $a[i][j]$ và chuyển qua xử lý ô trống tiếp theo.
- Nếu không tìm được giá trị c ta lùi lại ô trống trước đó.
- Thuật toán kết thúc thành công khi mọi ô trống đều được điền giá trị hợp lý.
- Thuật toán kết thúc vô nghiệm nếu ta quay lui về điểm xuất phát sau khi đã duyệt hết mọi khả năng.

Khởi trị: (i,j) là ô trống đầu tiên

while True:

if dòng $i > 8$: Thông báo nghiệm; return

if dòng $i < 0$: Thông báo vô nghiệm; return

if Tìm được một giá trị c cho ô (i,j) : Tiến 1 bước

else: Lùi 1 bước

Sơ đồ thuật toán quay lui

Tổ chức dữ liệu

Trước hết ta cài đặt kiểu dữ liệu Index quản lý các chỉ số của mỗi ô trong bảng Sudoku.

Index gồm hai chỉ số dòng (row) và cột (col) với các trị 1..9.

```
class Index {
public:
    int row, col;
    // constructors
    Index() { row = col = 0; }
    Index(int i, int j) { row = i; col = j; }
    Index(const Index &y) { row = y.row; col = y.col; }
    Set() { row = col = 0; }
    Set(int i, int j) { row = i; col = j; }
    Set(const Index &y) { row = y.row; col = y.col; }
```

```

    bool operator ==(Index y) { return row == y.row && col == y.col; }
    bool operator !=(Index y) { return !(*this == y); }
    Print(const char*msg = "") {
        cout << msg << "(" << row << ", " << col << ")";
    }
};

```

Sau khi đọc dữ liệu vào mảng 9×9 s, ta lần lượt duyệt các ô trống (lúc đầu chứa số 0). Với mỗi ô trống (i,j) ta tổ chức hai mảng Next[i][j] trỏ đến ô trống tiếp theo ô trống (i,j) và Pre[i][j] trỏ đến ô trống trước ô trống (i,j).

Hàm GetStart() tìm ô trống đầu tiên start để làm ô xuất phát cho quá trình duyệt.

```

Index start;
void GetStart() {
    // tìm điểm 0 đầu tiên start
    start.Set(-1,-1);
    for (int i = 0; i < MN; ++i) {
        for (int j = 0; j < MN; ++j) {
            if (s[i][j] == 0) {
                start.Set(i,j);
                return;
            } // if
        } // for j
    } // for i
}

```

Tổ chức các con trỏ trước và sau cho mỗi ô trống $s[i][j] = 0$

```

void Init() {
    GetStart(); // Tìm điểm 0 đầu tiên ghi vào start
    start.Print("\n Start");
    if (start.row < 0) {
        cout << "\n Correct Sudoku.";
        return;
    }
    // Tìm các điểm 0
    // Khởi trị cho Pred and Next
    Index x(-1,-1), y(start); // x là điểm trước, y là điểm sau
    for (int i = 0; i < MN; ++i) {
        for (int j = 0; j < MN; ++j) {
            if (s[i][j] == 0) {

```

```

        Pred[i][j] = x;
        x = Index(i,j);
        Next[y.row][y.col] = x;
        y = x;
    }
} // for i
} // for j
Next[y.row][y.col] = Index(MN,MN);
}

```

Hàm BTL: Backtracking

```

int BT(){ // Backtracking
    p = start;
    p.Print("\n Diem xuat phat: ");
    while(1) {
        //Print(s); p.Print("\n p : ");Go();
        if (p.row > 8) return 1;
        if (p.row < 0) return 0;
        SetVals(p, Find());
        p = (Vals(p) > 0) ? Next[p.row][p.col]
            : Pred[p.row][p.col];
    } // while
}

```

Hàm Find()

Hàm Find() trước hết đánh dấu các số đã ghi trên dòng si, cột sj và khối 3×3 chứa ô đang xét (si, sj) vào mảng một chiều c: c[i] = 0 cho biết số i chưa xuất hiện, ngược lại, c[i] = 1 cho biết số i đã xuất hiện trên dòng si hoặc cột sj hoặc trong khối chứa ô (si, sj). Sau đó duyệt lại c để tìm ô trống đầu tiên c[i] = 0. Nếu tìm được thì i chính là số cần điền cho ô (si, sj). Nếu không, hàm cho ra giá trị 0.

```

// Tim tri cho o s[p.row][p.col]
int Find() {
    memset(used,0,sizeof(used));
    // danh dau cac so da dung trong s
    for (int i = 0; i < MN; ++i)
        used[s[p.row][i]] = used[s[i][p.col]] = 1;
    // Dau khoi: (di,dj)
    int di = DD[p.row];

```

```

    int dj = DD[p.col];
    for (int i = 0; i < 3; ++i)
        for (int j = 0; j < 3; ++j)
            used[s[di+i][dj+j]] = 1;
    // tìm số đầu tiên chưa dùng
    for (int i = Vals(p)+1; i <= MN; ++i)
        if (!used[i]) return i;
    return 0;
}

```

trong đó: (DD[i], DD[j]) vị trí góc trên trái của khối 3×3 chứa ô (i, j).

```
DD = [0,0,0,3,3,3,6,6,6] // điểm đầu của box 3x3
```

Hàm Test có nhiệm vụ kiểm tra xem một lời giải có đáp ứng đầy đủ các yêu cầu của bảng số Sudoku không.

Ta sử dụng hai mảng Next và Pred để xác định các ô trống (chứa giá trị 0) đứng sau và đứng trước ô trống (i,j). Ta dùng hai ô (-1,-1) và (9,9) làm *lính canh* cho next và pred, cụ thể là:

Nếu (i,j) là ô trống cuối cùng thì Next[i][j] = (9,9)

Nếu (i,j) là ô trống đầu tiên thì pred[i][j] = (-1,-1)

Program

```

// SUDOKU
#include<bits/stdc++.h>
using namespace std;
const char * FN = "SUDOKU.INP";
const int MN = 9;
int s[MN][MN]; // cấu hình gốc
int used[10]; // các số đã dùng
// Điểm đầu các block
const int DD[] = {0,0,0,3,3,3,6,6,6};
class Index {
public:
    int row, col;
    // constructors
    Index() { row = col = 0; }
    Index(int i, int j) { row = i; col = j; }
    Index(const Index &y) { row = y.row; col = y.col; }
    Set() { row = col = 0; }
}

```

```

Set(int i, int j) { row = i; col = j; }
Set(const Index &y) { row = y.row; col = y.col; }
bool operator ==(Index y) { return row == y.row && col == y.col; }
bool operator !=(Index y) { return !(*this == y); }
Print(const char*msg = "") {
    cout << msg << "(" << row << ", " << col << ")";
}
};

Index p; // current position
Index start; // vị trí xuất phát tại điểm 0
Index Next[MN][MN], Pred[MN][MN];
inline void Go(const char * msg = " ? ") {
    cout << msg; fflush(stdin);
    if (cin.get() == '.') exit(0);
}
// Hien thi mang 1 chieu a[d..c]
void Print(int a[], const char *msg = "") {
    cout << msg;
    for (int i = 0; i < MN; ++i)
        cout << " " << a[i];
}
// Hien thi ma tran 2 chieu a[d..c][d..c]
void Print(int a[][MN], const char *msg = "") {
    cout << msg;
    for (int i = 0; i < MN; ++i)
        Print(a[i], "\n ");
}
void Input() { // doc cau hinh Sudoku s
    ifstream f(FN);
    if (f.fail()) {
        cerr << "\n unable open file " << FN;
        exit(0);
    }
    for (int i = 0; i < MN; ++i)
        for (int j = 0; j < MN; ++j)
            f >> s[i][j];
    f.close();
}

```



```

}
void GetStart() {
    // tìm điểm 0 đầu tiên start
    start.Set(-1,-1);
    for (int i = 0; i < MN; ++i)
        for (int j = 0; j < MN; ++j)
            if (s[i][j] == 0) { start.Set(i,j); return; }
}
inline int Vals(Index p) { return s[p.row][p.col]; }
inline void SetVals(Index p, int v) { s[p.row][p.col] = v; }
inline int Vals(int i, int j) { return s[i][j]; }
inline void SetVals(int i, int j, int v) { s[i][j] = v; }
// Tìm tri cho o s[p.row][p.col]
int Find() {
    memset(used,0,sizeof(used));
    for (int i = 0; i < MN; ++i) {
        used[s[p.row][i]] = used[s[i][p.col]] = 1;
    }
    // Dau khoi: (di,dj)
    int di = DD[p.row];
    int dj = DD[p.col];
    for (int i = 0; i < 3; ++i)
        for (int j = 0; j < 3; ++j)
            used[s[di+i][dj+j]] = 1;
    for (int i = Vals(p)+1; i <= MN; ++i)
        if (!used[i]) return i;
    return 0;
}
int TestRow(int i){
    memset(used, 0, sizeof(used));
    for (int j = 0; j < MN; ++j) {
        if (used[s[i][j]] > 0) {
            cout << " \n Error in row " << i+1;
            return 1;
        }
        used[s[i][j]] = 1;
    }
}

```

```

    for (int j = 1; j <= MN; ++j) {
        if (used[j] == 0) {
            cout << " \n Error in row " << i+1;
            return 1;
        }
    }
    return 0;
}

int TestColumn(int i) {
    memset(used, 0, sizeof(used));
    for (int j = 0; j < MN; ++j) {
        if (used[s[j][i]] > 0) {
            cout << " \n Error in column " << i+1;
            return 1;
        }
        used[s[j][i]] = 1;
    }
    for (int j = 1; j <= MN; ++j) {
        if (used[j] == 0) {
            cout << " \n Error in column " << i+1;
            return 1;
        }
    }
    return 0;
}

int TestBlock(int di, int dj) {
    memset(used, 0, sizeof(used));
    for (int i = 0; i < 3; ++i) {
        for (int j = 0; j < 3; ++j) {
            if (used[s[di+i][dj+j]] > 0) {
                cout << "\n Error in block "
                    << di+1 << " " << dj+1;
                return 1;
            } // if
            used[s[di+i][dj+j]] = 1;
        } // for j
    }
}

```

```

    } // for i
    for (int j = 1; j <= MN; ++j) {
        if (used[j] == 0) {
            cout << " \n Error in block "
                << di+1 << " " << dj+1;
            return 1;
        }
    }
    return 0;
}

int Test() {
    cout << "\n Testing...";
    int er = 0; // dem loi
    for (int i = 0; i < MN; ++i) {
        er += TestRow(i);
        er += TestColumn(i);
    }
    for (int i = 0; i < MN; i += 3)
        for (int j = 0; j < MN; j += 3)
            er += TestBlock(i,j);
    if (er == 0) cout << "\n Correct.";
    else cout << "\n Total error: " << er;
}

int BT(){ // Backtracking
    p = start;
    p.Print("\n Diem xuat phat: ");
    while(1) {
        if (p.row > 8) return 1;
        if (p.row < 0) return 0;
        SetVals(p, Find());
        p = (Vals(p) > 0) ? Next[p.row][p.col]
            : Pred[p.row][p.col];
    } // while
}

// To chuc cac tro truoc va sau cho moi s[i][j] = 0

```

```

void Init() {
    GetStart(); // Tim diem 0 dau tien ghi vao start
    start.Print("\n Start");
    if (start.row < 0) {
        cout << "\n Correct Sudoku.";
        return;
    }
    // Tim cac diem 0
    // Khoi tri cho Pred and Next
    Index x(-1,-1), y(start); // x la diem truoc, y la diem sau
    for (int i = 0; i < MN; ++i) {
        for (int j = 0; j < MN; ++j) {
            if (s[i][j] == 0) {
                Pred[i][j] = x;
                x = Index(i,j);
                Next[y.row][y.col] = x;
                y = x;
            }
        } // for i
    } // for j
    Next[y.row][y.col] = Index(MN,MN);
}

void Sudoku() {
    Input();
    Print(s, "\n Input: ");
    Init();
    BT();
    Print(s, "\n Result: \n");
    Test();
}

int main() {
    Sudoku();
    cout << endl << " T h e   E n d";
    return 0;
}

```

254 CHUYỀN TIN 11: CÁC CHUYỀN ĐỀ CHỌN LỌC VỚI NGÔN NGỮ C++

Dưới đây là ba test kèm theo đáp án. Lưu ý rằng có thể có nhiều đáp án cho một test.

Test 1	Test 2	Test 3
4 0 0 0 0 0 6 0 0	0 0 0 0 0 0 0 7 0	0 3 8 0 7 0 0 0 0
0 9 0 0 8 0 0 5 0	0 0 0 0 7 0 0 0 0	0 0 0 0 0 0 0 7 0
5 0 0 0 0 9 2 0 0	0 0 0 0 0 0 0 0 0	0 0 0 8 0 5 0 1 0
6 0 0 0 0 0 1 0 0	0 0 0 0 0 0 0 0 0	0 5 9 0 6 0 3 0 0
0 2 0 0 7 0 0 3 0	0 0 0 0 0 0 0 0 0	0 0 0 0 9 0 8 0 0
0 0 4 0 0 0 0 0 9	0 0 0 0 0 0 0 5 0	4 0 0 1 2 0 9 0 0
0 0 1 3 0 0 0 0 8	0 0 0 0 0 7 0 0 0	5 0 0 0 0 0 0 0 0
0 3 0 0 2 0 0 4 0	0 0 7 8 0 0 0 0 4	0 9 0 0 1 0 0 0 0
0 0 9 0 0 0 0 0 6	8 0 0 0 0 0 0 0 0	2 7 0 9 0 0 0 0 0

Đáp án Test 1	Đáp án Test 2	Đáp án Test 3
4 1 8 2 5 3 6 9 7	1 2 3 4 5 6 8 7 9	6 3 8 2 7 1 4 9 5
3 9 2 6 8 7 4 5 1	4 5 6 9 7 8 1 2 3	1 4 5 6 3 9 2 7 8
5 6 7 4 1 9 2 8 3	7 8 9 1 2 3 4 6 5	9 2 7 8 4 5 6 1 3
6 8 3 5 9 4 1 7 2	2 1 4 3 6 5 7 9 8	7 5 9 4 6 8 3 2 1
9 2 5 1 7 6 8 3 4	3 6 5 7 8 9 2 4 1	3 1 2 5 9 7 8 6 4
1 7 4 8 3 2 5 6 9	9 7 8 2 1 4 3 5 6	4 8 6 1 2 3 9 5 7
7 4 1 3 6 5 9 2 8	5 3 1 6 4 7 9 8 2	5 6 1 3 8 2 7 4 9
8 3 6 9 2 1 7 4 5	6 9 7 8 3 2 5 1 4	8 9 4 7 1 6 5 3 2
2 5 9 7 4 8 3 1 6	8 4 2 5 9 1 6 3 7	2 7 3 9 5 4 1 8 6

Auto Test

Bạn có thể run (chạy) tự động thông qua việc sinh dữ liệu nhờ hàm randrange như sau:

```
for each number k = 1..9
    ghi k vào ô ngẫu nhiên trong bản 9×9
```

Algorithm

```
void AutoRun() {
    srand(time(NULL));
    int i, j;
    while(true) {
        memset(s, 0, MN*MN*sizeof(int));
        for (int k = 1; k <= MN; ++k) {
```

```

do {
    i = rand() % 9; j = rand() % 9;
} while (s[i][j] > 0);
s[i][j] = k;
}
Print(s, "\n Input: ");
Init();
BT();
Print(s, "\n Result: \n");
Test();
Go();
}
}

```

Bạn cũng có thể viết thủ tục sau đây để sinh nhanh một cấu hình Sudoku chuẩn

1	2	3	4	5	6	7	8	9
4	5	6	7	8	9	1	2	3
7	8	9	1	2	3	4	5	6
2	3	4	5	6	7	8	9	1
5	6	7	8	9	1	2	3	4
8	9	1	2	3	4	5	6	7
3	4	5	6	7	8	9	1	2
6	7	8	9	1	2	3	4	5
9	1	2	3	4	5	6	7	8

```

// sinh cau hinh chuan
void GenNorm() {
    int d = 0;
    for(int i = 0; i < MN; ++i) {
        int c = d;
        for(int j = 0; j < MN; ++j) {
            c = c % 9 + 1;
            s[i][j] = c;
            d = (d+3) % 8;
            if (d == 0) d = 8;
        }
    }
}
}

```

Autotest2

Sau khi tạo cấu hình chuẩn, bạn xóa ngẫu nhiên một số ô bằng cách gán trị 0.

```
void AutoTest2(int limit = 50) {
    srand(time(NULL));
    while(true) {
        GenNorm(); // sinh cau hinh chuan
        // xoa ngau nhien mot so o
        for (int k = 1; k <= limit; ++k)
            s[rand() % 9][rand() % 9] = 0;
        Print(s, "\n Input: ");
        Init(); BT();
        Print(s, "\n Result: \n");
        Test(); Go();
    }
}
```

Program for Autorun

```
// SUDOKU : AutoRun
#include<bits/stdc++.h>
using namespace std;
const char * FN = "SUDOKU.INP";
. . .
void Sudoku() {
    Input();
    Print(s, "\n Input: ");
    Init(); BT();
    Print(s, "\n Result: \n");
    Test();
}
void AutoRun1() {
    srand(time(NULL));
    int i, j;
    while(true) {
        memset(s, 0, MN*MN*sizeof(int));
        for (int k = 1; k <= MN; ++k) {
            do {
                i = rand() % 9; j = rand() % 9;
            } while (s[i][j] > 0);
        }
    }
}
```

```

        s[i][j] = k;
    }
    Print(s, "\n Input: ");
    Init();
    BT();
    Print(s, "\n Result: \n");
    Test();
    Go();
}
}
// sinh cau hinh chuan
void GenNorm() {
    int d = 0;
    for(int i = 0; i < MN; ++i) {
        int c = d;
        for(int j = 0; j < MN; ++j) {
            c = c % 9 + 1;
            s[i][j] = c;
            d = (d+3) % 8;
            if (d == 0) d = 8;
        }
    }
}
}
void AutoRun2(int limit = 50) {
    srand(time(NULL));
    while(true) {
        GenNorm(); // sinh cau hinh chuan
        // xoa ngau nhien mot so o
        for (int k = 1; k <= limit; ++k)
            s[rand() % 9][rand() % 9] = 0;
        Print(s, "\n Input: ");
        Init();
        BT();
        Print(s, "\n Result: \n");
        Test();
        Go();
    }
}
}

```



```
int main() {
    // Sudoku();
    // AutoRun1();
    AutoRun2();
    cout << endl << " T h e   E n d";
    return 0;
}
```

Các biến thể của Sudoku

Wordoku

Sudoku có nhiều biến thể do nhiều fan tham gia phát triển. Một trong những biến thể khá thú vị của Sudoku là *Sudoku dạng chữ* với tên gọi *Wordoku* hoặc *Alphabetical Sudoku*. Trong trò chơi này, các chữ số 1..9 được thay bằng 9 chữ cái khác nhau. Ví dụ, với đề bài đã cho ban đầu ta có thể thay 9 chữ số bằng dãy chữ cái QWERTYUIO – một định danh bàn phím với 9 chữ cái đầu.

4	0	0	0	0	0	6	0	0
0	9	0	0	8	0	0	5	0
5	0	0	0	0	9	2	0	0
6	0	0	0	0	0	1	0	0
0	2	0	0	7	0	0	3	0
0	0	4	0	0	0	0	0	9
0	0	1	3	0	0	0	0	8
0	3	0	0	2	0	0	4	0
0	0	9	0	0	0	0	0	6

a) Đề bài dạng số

Các ô trống hiện ghi số 0.

4	1	8	2	5	3	6	9	7
3	9	2	6	8	7	4	5	1
5	6	7	4	1	9	2	8	3
6	8	3	5	9	4	1	7	2
9	2	5	1	7	6	8	3	4
1	7	4	8	3	2	5	6	9
7	4	1	3	6	5	9	2	8
8	3	6	9	2	1	7	4	5
2	5	9	7	4	8	3	1	6

b) Lời giải dạng số

R	*	*	*	*	*	Y	*	*
*	O	*	*	I	*	*	T	*
T	*	*	*	*	O	W	*	*
Y	*	*	*	*	*	Q	*	*
*	W	*	*	U	*	*	E	*
*	*	R	*	*	*	*	*	O
*	*	Q	E	*	*	*	*	I
*	E	*	*	W	*	*	R	*
*	*	O	*	*	*	*	*	Y

c) Đề bài QWERTYUIO

Các ô trống hiện ghi dấu *

R	Q	I	W	T	E	Y	O	U
E	O	W	Y	I	U	R	T	Q
T	Y	U	R	Q	O	W	I	E
Y	I	E	T	O	R	Q	U	W
O	W	T	Q	U	Y	I	E	R
Q	U	R	I	E	W	T	Y	O
U	R	Q	E	Y	T	O	W	I
I	E	Y	O	W	Q	U	R	T
W	T	O	U	R	I	E	Q	Y

d) Lời giải QWERTYUIO

Input File trong bài này sẽ gồm 10 dòng. Dòng đầu chứa *mẫu* (sample) gồm 9 ký tự khác nhau. Các dòng còn lại là bảng sudoku với các chữ cái lấy từ mẫu, các dấu sao * được dùng để thể hiện ô trống.

```
QWERTYUIO
R*****Y**
*O**I**T*
T****OW**
Y*****Q**
*W**U**E*
**R*****O
**QE****I
*E**W**R*
**O*****Y
```

Wordoku khá dễ giải. Bạn chỉ cần đọc mẫu và thực hiện chuyển đổi các chữ cái sang số. Sau đó bạn giải sudoku số. Khi hiển thị kết quả bạn nhớ chuyển ngược số sang chữ là xong.

Program Wordoku

```
/*-----
WORDDOKU.CPP
-----*/
#include<bits/stdc++.h>
using namespace std;
const char * FN = "WORDOKU.INP";
const int MN = 9;
int s[MN][MN]; // cau hinh goc
int used[10]; // cac so da dung
// Diem dau cac block
const int DD[] = {0,0,0,3,3,3,6,6,6};
string w;
class Index { . . . };
Index p; // current position
Index start; // vi tri xuat phat tai diem 0
Index Next[MN][MN], Pred[MN][MN];
inline void Go(const char * msg = " ? ") { . . . }
// Hien thi mang 1 chieu a[d..c]
void Print(int a[], const char *msg = "") { . . . }
// Hien thi ma tran 2 chieu a[d..c][d..c]
```

260 CHUYÊN TIN 11: CÁC CHUYÊN ĐỀ CHỌN LỌC VỚI NGÔN NGỮ C++

```

void Print(int a[][MN], const char *msg = "") { . . . }
void GetStart() { . . . }
. . .
int TestRow(int i) { . . . }

```

```

int TestColum(int i) { . . . }
int TestBlock(int di, int dj) { . . . }
int Test() { . . . }
    int BT() { . . . }
    void Init() { . . . }
void Input() { // doc cau hinh Sudoku s
    ifstream f(FN);
    if (f.fail()) {
        cerr << "\n unable open file " << FN;
        exit(0);
    }
    string v;
    getline(f, w);
    w = "*" + w;
    cout << endl << w;
    for (int i = 0; i < MN; ++i) {
        // doc va chuyen doi dong i sang so
        getline(f, v);
        cout << endl << v;
        for(int j = 0; j < v.length(); ++j) {
            s[i][j] = w.find(v[j]);
        }
    }
    f.close();
}

void Wordoku() {
    Input();
    Print(s, "\n Input: ");
    Init();
    BT();
    Print(s, "\n Digit Result: \n");
}

```

```

    Test();
    // chuyen nguoc lai dang chu
    cout << "\n Result: ";
    for (int i = 0; i < MN; ++i) {
        cout << endl << " ";
        for(int j = 0; j < MN; ++j) {
            cout << " " << w[s[i][j]];
        }
    }
}

int main() {
    Wordoku();
    cout << endl << " T h e   E n d";
    return 0;
}

```

Output

```

*QWERTYUIO
R*****Y**
*O**I**T*
T****OW**
Y*****Q**
*W**U**E*
**R*****O
**QE*****I
*E**W**R*
**O*****Y

```

Input:

```

4 0 0 0 0 0 6 0 0
0 9 0 0 8 0 0 5 0
5 0 0 0 0 9 2 0 0
6 0 0 0 0 0 1 0 0
0 2 0 0 7 0 0 3 0
0 0 4 0 0 0 0 0 9
0 0 1 3 0 0 0 0 8
0 3 0 0 2 0 0 4 0
0 0 9 0 0 0 0 0 6

```

```

Start(0, 1)
Diem xuat phat: (0, 1)
Digit Result:
4 1 8 2 5 3 6 9 7
3 9 2 6 8 7 4 5 1
5 6 7 4 1 9 2 8 3
6 8 3 5 9 4 1 7 2
9 2 5 1 7 6 8 3 4
1 7 4 8 3 2 5 6 9
7 4 1 3 6 5 9 2 8
8 3 6 9 2 1 7 4 5
2 5 9 7 4 8 3 1 6
Correct.
Result:
R Q I W T E Y O U
E O W Y I U R T Q
T Y U R Q O W I E
Y I E T O R Q U W
O W T Q U Y I E R
Q U R I E W T Y O
U R Q E Y T O W I
I E Y O W Q U R T
W T O U R I E Q Y
T h e   E n d

```

New Sudoka

Chúng ta giải một bài Sudoku dạng chữ với giả thiết đơn giản sau đây: Cho trước một keyword w gồm 9 chữ cái. Yêu cầu xây dựng một bảng Sudoku chứa 9 chữ cái này trên đường chéo chính. Ví dụ, với keyword $w = \text{NEWSUDOKA}$ chương trình cho ta một lời giải như ở bảng dưới.

Bài này cũng dễ giải.

Dữ liệu vào trong bài này chỉ duy nhất một dòng 9 chữ cái w như trên, trong đó có thể có các chữ cái trùng lặp.

Việc đầu tiên là khởi trị bảng Sudoku số s chứa toàn 0, riêng đường chéo chính ta gán $s[i][i] = w[i]$, $1 \leq i \leq 9$. Đó là vị trí các chữ cái trong keyword w .

N	S	U	E	W	O	D	A	K
D	E	O	N	K	A	W	U	S
K	A	W	U	D	S	N	E	O
E	N	D	S	O	K	U	W	A
W	O	S	A	U	N	K	D	E
U	K	A	W	E	D	S	O	N
S	W	E	K	A	U	O	N	D
O	U	N	D	S	E	A	K	W
A	D	K	O	N	W	E	S	A

Một lời giải NEWSUDOKA

Program

```

/*-----
NEWSUDOKA
-----*/
#include<bits/stdc++.h>
using namespace std;
const char * FN = "SUDOKA.INP";
. . .
void Sudoka() { // doc cau hinh Sudoku s
    ifstream f(FN);
    if (f.fail()) {
        cerr << "\n unable open file " << FN;
        exit(0);
    }
    while(getline(f, w)) {
        cout << endl << w;
        if (w == "") break;
        // chuyen w sang s
        memset(s, 0, MN*MN*sizeof(int)); // all 0
        // dien tri cho duong cheo chinh
        for (int i = 0; i < w.length(); ++i) {
            s[i][i] = i+1;
        }
        Print(s, "\n Input: \n");
        BT();
        Print(s, "\n Digit Result: \n");
    }
}

```

```

    Test();
    // chuyen nguoc lai dang chu
    cout << "\n Result: ";
    w = "0"+w;
    for (int i = 0; i < MN; ++i) {
        cout << endl << " ";
        for(int j = 0; j < MN; ++j) {
            cout << " " << w[s[i][j]];
        } // for j
    } // for i
    Go();
} // while
f.close();
}
int main() {
    Sudoka();
    cout << endl << " T h e   E n d";
    return 0;
}

```

Dưới đây liệt kê một số từ khóa cho bài NewSudoka.

```

File SUDOKA.INP
QWERTYUIO (Hàng phím chữ đầu tiên)
NEWSUDOKA
LAP*TRINH (Lập trình)
VIETNAMHP (Việt Nam Hạnh phúc)
TOM*JER2Y (Tom and Jerry)
CHUYENGIA (Chuyên gia)
ROMANTICS (Lãng mạn, phong cách Rome)
KEYBOARDS (bàn phím)
PROGWITHC (Lập trình với ngôn ngữ C)
ABCPYTHON (Nhập môn Python)
MCDIEUHAN (MC Diệu Hân)
BORLANDSC (Công ty Phần mềm Borland)

```

Game Sudoku

Ta xây dựng một phần mềm tổng hợp các biến thể của Sudoku. Người dùng có thể tự chọn một trong các phương án sau:

✂ Sudoku kinh điển

✂ Sinh tự động các đề Sudoku và giải đề

✂ Wordoku

✂ New Sudoka

Bạn cần chuẩn bị trước hai file dữ liệu sau đây:

✂ SUDOKU.INP

✂ SUDOKU.WRD

SUDOKU.INP	SUDOKU.WRD
3	QWERTYUIO (Hàng phím chữ đầu tiên)
0 3 8 0 7 0 0 0 0	NEWSUDOKA
0 0 0 0 0 0 0 7 0	LAP*TRINH (Lập trình)
0 0 0 8 0 5 0 1 0	VIETNAMHP (Việt Nam Hạnh phúc)
0 5 9 0 6 0 3 0 0	TOM*JER2Y (Tom and Jerry)
0 0 0 0 9 0 8 0 0	CHUYENGIA (Chuyên gia)
4 0 0 1 2 0 9 0 0	ROMANTICS (Lãng mạn, phong cách Rome)
5 0 0 0 0 0 0 0 0	KEYBOARDS (bàn phím)
0 9 0 0 1 0 0 0 0	PROGWITHC (Lập trình với ngôn ngữ C)
2 7 0 9 0 0 0 0 0	ABCPYTHON (Nhập môn Python)
... (cấu hình 2)	MCDIEUHAN (MC Diệu Hân)
... (cấu hình 3)	BORLANDSC (Công ty Phần mềm Borland)
	HANOI*DEP (Hà Nội đẹp)

Chúng ta sẽ thiết kế game thực hiện các chức năng sau:

1. Auto Run 1
2. Auto Run 2
3. Sudoku
4. Sudoka
5. Wordoku

Riêng với Wordoku sẽ có chút thay đổi. Cụ thể là thủ tục này sẽ đọc từng dòng mẫu trong file SUDOKU.WRD để sinh ngẫu nhiên các bảng chứa một số chữ cái trong mẫu, các ô còn lại thì chứa 0. Sau khi giải được cấu hình thì hiển thị kết quả theo mẫu.

Program

```
/*-----
Game Sudoku
-----*/
#include<bits/stdc++.h>
```



```

using namespace std;
const char * sudoku = "SUDOKU.INP";
const char * sudoword = "SUDOKU.WRD";
const int MN = 9;
int s[MN][MN]; // cau hinh goc
int used[10]; // cac so da dung
// Diem dau cac block
const int DD[] = {0,0,0,3,3,3,6,6,6};
class Index {
public:
    int row, col;
    Index() { row = col = 0; }
    Index(int i, int j) { row = i; col = j; }
    Index(const Index &y) { row = y.row; col = y.col; }
    Set() { row = col = 0; }
    Set(int i, int j) { row = i; col = j; }
    Set(const Index &y) { row = y.row; col = y.col; }
    bool operator ==(Index y) { return row == y.row && col == y.col; }
    bool operator !=(Index y) { return !(*this == y); }
    Print(const char*msg = "") {
        cout << msg << "(" << row << ", " << col << ")";
    }
};
Index p; // current position
Index start; // vi tri xuat phat tai diem 0
Index Next[MN][MN], Pred[MN][MN];
inline char Wait(const char * msg
    = "\n Press dot key [.] to the Menu: ") {
    cout << msg; fflush(stdin);
    return cin.get();
}
// Hien thi mang 1 chieu a[d..c]
void Print(int a[], const char *msg = "") {
    cout << msg;
    for (int i = 0; i < MN; ++i)
        cout << " " << a[i];
}

```

```

// Hien thi ma tran 2 chieu a[d..c][d..c]
void Print(int a[][MN], const char *msg = "") {
    cout << msg;
    for (int i = 0; i < MN; ++i) Print(a[i], "\n ");
}

// Hien thi ma tran 2 chieu a[d..c][d..c] theo mai w
void Print(const string &w, int a[][MN], const char *msg = "") {
    cout << msg;
    for (int i = 0; i < MN; ++i) {
        cout << "\n ";
        for (int j = 0; j < MN; ++j)
            cout << " " << w[a[i][j]];
    }
}

inline int Vals(Index p) { return s[p.row][p.col]; }
inline void SetVals(Index p, int v) { s[p.row][p.col] = v; }
inline int Vals(int i, int j) { return s[i][j]; }
inline void SetVals(int i, int j, int v) { s[i][j] = v; }

// Tim tri cho o s[p.row][p.col]
int Find(int row, int col, int from = 1) {
    memset(used, 0, sizeof(used));
    for (int i = 0; i < MN; ++i) {
        used[s[row][i]] = used[s[i][col]] = 1;
    }
    // Dau khoi: (di,dj)
    int di = DD[row];
    int dj = DD[col];
    for (int i = 0; i < 3; ++i)
        for (int j = 0; j < 3; ++j)
            used[s[di+i][dj+j]] = 1;
    // Khi test i: 0..9;
    // khi goi BT i: vals(row, col)+1..0
    if (from == 1) from = Vals(row, col);
    for (int i = from+1; i <= MN; ++i)
        if (!used[i]) return i;
    return 0;
}

```

```

int Test() {
    int er = 0; // dem loi
    for (int i = 0; i < MN; ++i) {
        for (int j = 0; j < MN; ++j) {
            if (Find(i,j,0) > 0) {
                ++er;
                cout << "\n Error in cell " << i+1 << " " << j+1;
            }
        }
    }
    if (er == 0) cout << "\n Correct.";
    else cout << "\n Total error: " << er;
}

void GetStart() {
    // tim diem 0 dau tien start
    start.Set(-1,-1);
    for (int i = 0; i < MN; ++i) {
        for (int j = 0; j < MN; ++j) {
            if (s[i][j] == 0) {
                start.Set(i,j);
                return;
            } // if
        } // for j
    } // for i
}

// To chuc cac tro truoc va sau cho moi s[i][j] = 0
void Init() {
    GetStart(); // Tim diem 0 dau tien ghi vao start
    // start.Print("\n Start");
    if (start.row < 0) {
        cout << "\n Correct Sudoku.";
        return;
    }
    // Tim cac diem 0
    // Khoi tri cho Pred and Next
    Index x(-1,-1), y(start); // x la diem truoc, y diem sau
    for (int i = 0; i < MN; ++i) {

```

```

        for (int j = 0; j < MN; ++j) {
            if (s[i][j] == 0) {
                Pred[i][j] = x;
                x = Index(i,j);
                Next[y.row][y.col] = x;
                y = x;
            }
        } // for i
    } // for j
    Next[y.row][y.col] = Index(MN,MN);
}

int BT(){ // Backtracking
    Init();
    // cout << "\n Backtracking...";
    p = start;
    // p.Print("\n Diem xuat phat: ");
    while(1) {
        if (p.row > 8) return 1;
        if (p.row < 0) return 0;
        SetVals(p, Find(p.row, p.col));
        p = (Vals(p) > 0) ? Next[p.row][p.col]
            : Pred[p.row][p.col];
    } // while
}

void Exe(bool prt = 1) {
    if(prt) Print(s,"\n Input: ");
    BT();
    if(prt) Print(s,"\n Digit Result: ");
    Test();
}

void AutoRun1() {
    srand(time(NULL));
    int i, j;
    while(true) {
        memset(s, 0, MN*MN*sizeof(int));
        for (int k = 1; k <= MN; ++k) {
            do {

```

```

        i = rand() % 9; j = rand() % 9;
    } while (s[i][j] > 0);
    s[i][j] = k;
}
Exe();
if (Wait() == '.') return;
}
}
// sinh cau hinh chuan
void GenNorm() {
    int d = 0;
    for(int i = 0; i < MN; ++i) {
        int c = d;
        for(int j = 0; j < MN; ++j) {
            c = c % 9 + 1;
            s[i][j] = c;
            d = (d+3) % 8;
            if (d == 0) d = 8;
        }
    }
}
}
void AutoRun2(int limit = 50) {
    srand(time(NULL));
    while(true) {
        GenNorm(); // sinh cau hinh chuan
        // xoa ngau nhien mot so o
        for (int k = 1; k <= limit; ++k) {
            s[rand() % 9][rand() % 9] = 0;
        }
        Exe();
        if (Wait() == '.') return;
    }
}
void Sudoku() {
    ifstream f(sudoku);
    if (!f) {
        cerr << "\n unable open file " << sudoku;
    }
}

```

```

        exit(0);
    }
    int q;
    f >> q;
    cout << "\n " << q << " tests";
    for(int t = 1; t <= q; ++t) {
        for (int i = 0; i < MN; ++i)
            for(int j = 0; j < MN; ++j)
                f >> s[i][j];
        Exe();
        if (Wait() == '.') break;
    }
    f.close();
}

// xoa cac BL
string Trim(const string & s) {
    string w = "";
    for(auto c:s)
        if (c != 32) w += c;
    return w;
}

void Sudoka() { // doc cau hinh Sudoku s
    ifstream f(sudoword);
    if (f.fail()) {
        cerr << "\n unable open file " << sudoword;
        exit(0);
    }
    string w;
    while(getline(f, w)) {
        w = Trim(w);
        if( w == "") return;
        w = w.substr(0,9); // chi lay 9 chu cai dau
        cout << endl << w;
        if (w == "") break;
        // chuyen w sang dang so s
        memset(s, 0, MN*MN*sizeof(int)); // all 0
        // dien tri tren duong cheo chinh

```

```

        for (int i = 0; i < w.length(); ++i)
            s[i][i] = i+1;
        Exe(0);
        // chuyen nguoc lai dang chu
        w = "*" + w;
        Print(w, s, "\n Result: ");
        if (Wait() == '.') return;
    } // while
    f.close();
}

void Wordoku() {
    ifstream f(sudoword);
    if (f.fail()) {
        cerr << "\n unable open file " << sudoword;
        exit(0);
    }
    string w;
    while(getline(f, w)) {
        w = Trim(w);
        if( w == "") return;
        w = w.substr(0,9); // chi lay 9 chu cai dau
        cout << endl << w;
        w = "*" + w;
        // sinh ngau nhien bang so
        srand(time(NULL));
        int limit = 50;
        GenNorm(); // sinh cau hinh chuan
        // xoa ngau nhien mot so o
        for (int k = 1; k <= limit; ++k)
            s[rand() % 9][rand() % 9] = 0;
        Print(w, s, "\n Init: "); // hien thi theo chu
        Exe(0);
        Print(w, s, "\n Result: "); // hien thi theo chu
        if (Wait() == '.') return;
    } // while
    f.close();
}

```

```

void Run() {
    string command = "012345";
    cout << "\n S U D O K U   G A M E ";
    int com = 1;
    while (com != 0) {
        cout << "\n 0. Exit";
        cout << "\n 1. Auto Run 1";
        cout << "\n 2. Auto Run 2";
        cout << "\n 3. Sudoku";
        cout << "\n 4. Sudoka";
        cout << "\n 5. Wordoku";
        while (true) {
            cout << "\n Select 0..5: ";
            fflush(stdin);
            char c = cin.get();
            com = command.find(c);
            if (com >= 0 && com <= 5) break;
        } // while
        switch (com) {
            case 0: return;
            case 1: cout << "\n 1. Auto Test 1";
                    AutoRun1(); break;
            case 2: cout << "\n 2. Auto Test 2";
                    AutoRun2(); break;
            case 3: cout << "\n 3. Sudoku";
                    Sudoku(); break;
            case 4: cout << "\n 4. Sudoka";
                    Sudoka(); break;
            case 5: cout << "\n 5. Wordoku";
                    Wordoku(); break;
        } // swith
    }
} // Run

int main() {
    Run();
    cout << endl << " T h e   E n d";
    return 0;
}

```


Output

S U D O K U G A M E

0. Exit

1. Auto Run 1

2. Auto Run 2

3. Sudoku

4. Sudoka

5. Wordoku

Select 0..5: 1

1. Auto Test 1

Input:

```
0 0 0 0 0 0 0 0 0
0 0 2 0 0 0 4 0 7
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 9 3 0
0 0 8 0 0 0 5 0 0
0 0 0 0 0 0 0 6 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 1 0 0 0
0 0 0 0 0 0 0 0 0
```

Digit Result:

```
1 3 4 2 5 7 6 8 9
5 6 2 3 8 9 4 1 7
7 8 9 1 4 6 2 5 3
2 1 5 6 7 4 9 3 8
6 4 8 9 1 3 5 7 2
3 9 7 5 2 8 1 6 4
4 2 1 7 3 5 8 9 6
8 7 6 4 9 1 3 2 5
9 5 3 8 6 2 7 4 1
```

Correct.

Press dot key [.] to the Menu: .

0. Exit

1. Auto Run 1

2. Auto Run 2

3. Sudoku

4. Sudoka

5. Wordoku

Select 0..5: 2

2. Auto Test 2

Input:

0 0 3 0 0 6 7 0 9

0 0 6 0 8 9 0 0 3

0 0 0 1 0 3 0 5 0

2 0 4 5 6 7 8 0 0

0 0 7 8 0 0 2 0 0

8 9 1 0 3 4 5 0 0

3 4 0 0 7 8 9 0 2

0 7 0 0 1 2 3 4 5

0 0 2 0 0 0 6 7 0

Digit Result:

1 2 3 4 5 6 7 8 9

4 5 6 7 8 9 1 2 3

7 8 9 1 2 3 4 5 6

2 3 4 5 6 7 8 9 1

5 6 7 8 9 1 2 3 4

8 9 1 2 3 4 5 6 7

3 4 5 6 7 8 9 1 2

6 7 8 9 1 2 3 4 5

9 1 2 3 4 5 6 7 8

Correct.

Press dot key [.] to the Menu: .

0. Exit

1. Auto Run 1

2. Auto Run 2

3. Sudoku

4. Sudoka

5. Wordoku

Select 0..5: 3

3. Sudoku

. . .

Correct.

0. Exit

1. Auto Run 1

2. Auto Run 2

3. Sudoku

4. Sudoka

5. Wordoku

Select 0..5: 4

4. Sudoka

QWERTYUIO

Correct.

Result:

```
Q R T W E U Y O I
Y W U Q I O E R T
I O E T Y R Q W U
W Q Y R U I O T E
E U R O T Q I Y W
T I O E W Y R U Q
R T Q I O W U E Y
O Y W U Q E T I R
U E I Y R T W Q O
```

Press dot key [.] to the Menu:

NEWSUDOKA

Correct.

Result:

```
N S U E W O D A K
D E O N K A W S U
K A W U D S N E O
E N D S O K A U W
W O S A U N K D E
U K A W E D S O N
S U N K A E O W D
A D E O N W U K S
O W K D S U E N A
```

Press dot key [.] to the Menu:

LAP*TRINH

Correct.

Result:

```
L * T A P I R H N
R A I L N H P * T
N H P T R * L A I
A L R * I N H T P
P I * H T L N R A
T N H P A R * I L
* T L N H A I P R
H R A I L P T N *
I P N R * T A L H
```

Press dot key [.] to the Menu: .

0. Exit

1. Auto Run 1

2. Auto Run 2

3. Sudoku

4. Sudoka

5. Wordoku

Select 0..5: 5

5. Wordoku

QWERTYUIO

Init:

```
* * * * T Y U I O
* T Y U * * * * *
U I O Q W E R T Y
W * R * Y U I O Q
T Y U * O Q * E *
I O Q * * R T * *
E R * Y * I * * W
Y U * * * W E R T
O Q * E R T Y * *
```

Correct.

Result:

```
Q W E R T Y U I O
R T Y U I O Q W E
U I O Q W E R T Y
W E R T Y U I O Q
```

278 CHUYÊN TIN 11: CÁC CHUYÊN ĐỀ CHỌN LỌC VỚI NGÔN NGỮ C++

T Y U I O Q W E R

I O Q W E R T Y U

E R T Y U I O Q W

Y U I O Q W E R T

O Q W E R T Y U I

Press dot key [.] to the Menu:

0. Exit

1. Auto Run 1

2. Auto Run 2

3. Sudoku

4. Sudoka

5. Wordoku

Select 0..5: 0

T h e E n d

TÀI LIỆU THAM KHẢO

- [1] Bùi Việt Hà, *Nhập môn thuật toán*, Nxb. Đại học Quốc gia Hà Nội, 2021.
- [2] Nguyễn Xuân Huy, *Sáng tạo trong thuật toán và lập trình, T1 – T8*, Nxb. Thông tin và Truyền thông, 2010 - 2022.
- [3] Nguyễn Xuân Huy, *Chuyên tin 10: Các chuyên đề chọn lọc với ngôn ngữ C++ (Lý thuyết, bài tập và lời giải)*, Nxb. Thông tin và Truyền thông, 2024.
- [4] Nguyễn Xuân Huy, *Sáng tạo trong thuật toán và lập trình với Python* (tuyển các bài toán tin nâng cao cho học sinh và sinh viên giỏi), Nxb. Đại học Quốc gia Hà Nội, 2021.
- [5] Crandall Richard, Pomerance Carl, *Prime Numbers: A Computational Perspective* (2nd ed.), Springer, p. 121, 2005, ISBN 9780387252827.
- [6] Ellis Horowitz, Sartal Sahni, *Fundamentals of Computer Algorithms*, PITMAN, 1978.
- [7] Donald E. Knuth, *The Art of Computer Programming*, Addison Wesley PC, T1, 2, 3, 1974.
- [8] Kenneth H. Rosen, *Elementary Number Theory And Its Applications* (3rd ed.), Addison Wesley PC, 1992.
- [9] <https://stats.ioinformatics.org/>
- [10] <https://icpc.global/regionals/abouticpc>

CHUYÊN TIN 11

CÁC CHUYÊN ĐỀ CHỌN LỌC VỚI NGÔN NGỮ C++
LÝ THUYẾT, BÀI TẬP VÀ LỜI GIẢI

Chịu trách nhiệm xuất bản, nội dung

Giám đốc - Tổng Biên tập: TS. TRẦN CHÍ ĐẠT

Chịu trách nhiệm bản thảo

Phó Giám đốc - Phó Tổng Biên tập: Ngô Thị Mỹ Hạnh

Biên tập nội dung: Nguyễn Long Biên - Trịnh Thị Thu Châu

Biên tập sách điện tử: Nguyễn Long Biên - Trịnh Thị Thu Châu

Trình bày sách: Trịnh Thị Thu Châu

Thiết kế bìa: Nguyễn Thị Kim Phúc

NHÀ XUẤT BẢN THÔNG TIN VÀ TRUYỀN THÔNG

Website: www.nxbthongtintruyenthong.vn; book365.vn; ebook365.vn; ebook.gov.vn

Trụ sở: Tầng 6, Tòa nhà Cục Tần số vô tuyến điện - 115 Trần Duy Hưng, phường Yên Hòa, TP. Hà Nội

Điện thoại: 024.35772143

Điện thoại Phát hành: 024.35772138

E-mail: nxb.tttt@mst.gov.vn

Chi nhánh Nhà xuất bản tại TP. Hồ Chí Minh:

Địa chỉ: 27 Nguyễn Bình Khiêm, phường Sài Gòn, TP. Hồ Chí Minh

Điện thoại: 028.35127750/51

E-mail: cnsq.nxbtttt@mst.gov.vn

Chi nhánh Nhà xuất bản tại miền Trung - Tây Nguyên:

Địa chỉ: 42 Trần Quốc Toản, phường Hải Châu, TP. Đà Nẵng

46 Y Jut, phường Buôn Ma Thuột, tỉnh Đắk Lắk.

Điện thoại: 0236.3897467/0262.3808088

Fax: 0236.3843359

E-mail: cndn.nxbtttt@mst.gov.vn

CHUYÊN TIN 11

CÁC CHUYÊN ĐỀ CHỌN LỌC VỚI NGÔN NGỮ C++
LÝ THUYẾT, BÀI TẬP VÀ LỜI GIẢI

ISBN: 978-604-45-1537-3



9 786044 515373



8 935217 121182

Giá: 155.000đ